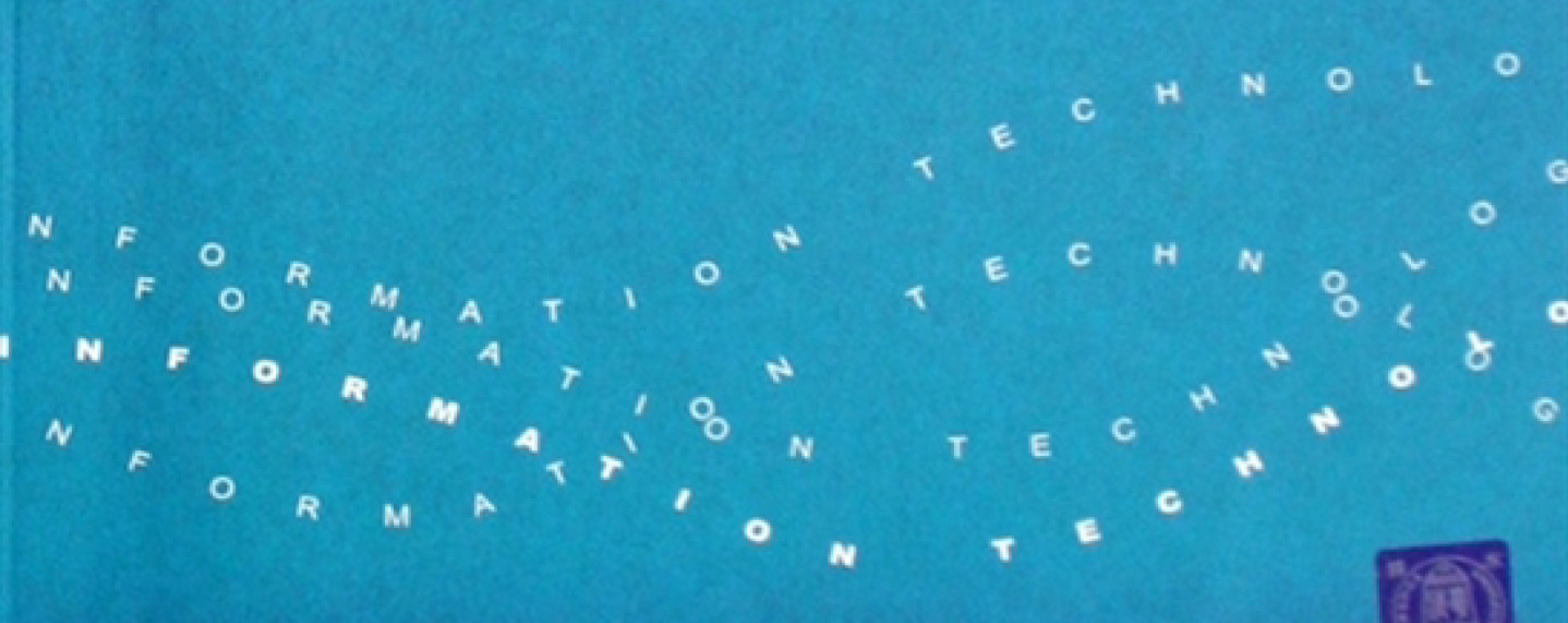
高等院校信息技术规划教材

Web应用开发技术

高 屹 齐东元 李 雷 编著



清华大学出版社

高等院校信息技术规划教材

Web应用开发技术

高　屹　齐东元　李　雷　编著

清华大学出版社

北　京

内 容 简 介

本书以一个完整的应用实例"畅想网络学院"为背景，系统地介绍了采用 ASP. NET 2.0 技术进行 Web 应用开发的相关内容，包括 Web 应用基础、ASP. NET 控件、构建网站、应用 ADO. NET 编程和 Web 数据访问等。

通过本书，读者能够学会在 Microsoft Visual Studio 2005 开发环境下，基于 Web 的网络应用程序的开发，以及最新的 Web 应用程序开发技术，掌握实际、有效的编程技巧，为实用系统的开发打下良好的基础。

本书作者长期从事计算机应用系统，特别是基于 Web 应用系统的开发工作，具有丰富的实际工作经验。本书的特点是理论与实践并重，既适合计算机相关专业的本、专科学生作为教材来使用，也可供广大 ASP. NET 开发人员和计算机软件爱好者学习参考。

图书在版编目（CIP）数据

Web 应用开发技术/高屹，齐东元，李雷编著．—北京：清华大学出版社，2008.7
（高等院校信息技术规划教材）
ISBN 978-7-302-17671-8

Ⅰ．W…　Ⅱ．①高…②齐…③李…　Ⅲ．主页制作—程序设计—高等学校—教材
Ⅳ．TP393.092

中国版本图书馆 CIP 数据核字(2008)第 079160 号

责任编辑：袁勤勇　李玮琪
责任校对：焦丽丽
责任印制：

出版发行：清华大学出版社　　**地　　址**：北京清华大学学研大厦 A 座
http://www. tup. com. cn　　**邮　　编**：100084
社 总 机：010-62770175　　**邮　　购**：010-62786544
投稿与读者服务：010-62776969，c-service@tup. tsinghua. edu. cn
质 量 反 馈：010-62772015，zhiliang@tup. tsinghua. edu. cn
印 刷 者：
装 订 者：
经　　销：全国新华书店
开　　本：185×260　**印　张**：20.5　　**字　数**：474 千字
版　　次：2008 年 7 月第 1 版　　**印　次**：2008 年 7 月第 1 次印刷
印　　数：1～0000
定　　价：0.00 元

本书如存在文字不清、漏印、缺页、倒页、脱页等印装质量问题，请与清华大学出版社出版部联系调换。联系电话：010-62770177 转 3103　　产品编号：027122-01

编委会名单

序 Preface

在科教兴国方针的指引下，我国高等教育进入了一个新的历史发展时期，招生规模和在校生数量都有了大幅度的增长。我们在进行着世界上规模最大的高等教育。与此同时，对于高等教育的研究和认识也在不断深化。高等学校要明确自己的办学方向和办学特色，这既是不断提高高等教育水平的必然要求，更是高校不断发展和壮大必须首先考虑的问题。

教育部领导明确提出，高等教育应多元化，高等院校应实施分类分层次教学，这是高等教育大众化的必然结果，也是市场对人才需求的客观规律所致。因此要有相当部分的高等院校致力于培养应用型人才。此类院校在计算机教学中如何实现自己的培养目标，如何选择适用的应用型教材，已成为十分重要和迫切的任务。应用型人才的培养不能简单照搬研究型人才的培养模式，要在丰富的实践基础上认真总结，摸索新形势下的教学规律，在此基础上设计相关课程、改进教学方法，同时编写与之相适应的应用型教材。这一工作是非常艰巨的，也是非常有意义的。

在清华大学出版社的大力支持和配合下，于2003年成立了应用型教材编委会。编委会汇集了众多高等院校的实践经验，并经过集中讨论和专家评审，遴选了一批优秀教材，希望能够通过这套教材的出版和使用，促进应用型人才培养的实践发展，为建立新的人才培养模式作出贡献。

我们编写应用型教材的主要出发点是：

1. 适应新形势下教育部"质量工程"对高等教育的要求及市场对应用型人才的需求。

2. 计算机科学技术和信息技术发展迅速，教材内容和教学方式应与之相适应，适时地进行更新和改进。

3. 教育技术的发展对教材建设提出了更高的要求，教材将呈现

出纸介质出版物、电子课件及网络学习环境等相互配合的立体化形态。

4. 根据不同的专业要求,突出应用,使理论与实践更加紧密结合。

以此为目标,我们将努力编写一套全新的、有实用价值的应用型计算机教材。经过参编教师的努力,第一批教材已经面世。教材将滚动式地不断更新、修正、提高,逐渐树立起自己的品牌。希望使用本系列教材的广大师生能对我们的教材提出宝贵的意见,共同建设具有应用型特色的精品教材。

朱　敏

前言 Foreword

随着 Internet 的普及，人们已经无法满足于在网上只浏览传统的静态内容。当前主流的网站基本都是以后台的数据库为基础，根据用户的请求，动态地生成网页内容。这样既能满足用户丰富的需求，网站的维护也更加方便。

当前，动态网站的开发技术主要有互相竞争的两大阵营：采用 .NET 技术的微软阵营和采用 JavaEE 体系结构的 Java 阵营。这两个阵营所采用的技术各有优势、相互竞争、共同发展。它们虽然采用不同的编程语言，但在体系结构上又有很大的相似性。

在微软阵营中，使用 ASP.NET 是当前进行 Web 应用开发的主要手段。自从微软推出 .NET 战略以来，ASP 技术一直是其重要组成部分。ASP.NET 不是 ASP 的简单升级，可以说是体系结构的全新换代。与 ASP 相比，使用 ASP.NET 开发动态 Web 应用更加简单、快捷和高效，因而受到广大程序员的喜爱。

本书在技术上没有倾向性，但选择 ASP.NET 为基础，介绍 Web 应用开发的一般方法及一些具有普遍意义的内容。

本书共有 13 章，可分为 3 个部分。

第 1 部分主要介绍一些与 Web 应用开发相关的基础知识，包括：

第 1 章　Web 应用基础，主要介绍几种 Web 应用开发所必须具备的背景知识，包括 HTML 超文本标记语言、CSS 层叠样式表和 JavaScript 脚本语言等。

第 2 章　ASP.NET 开发入门，主要介绍 Visual Studio 2005 的安装及其集成开发环境的使用等基础内容。

第 3 章　C# 语言基础，概要地介绍微软随 .NET 一起推出的一种新型编程语言 C#，本书所有示例的服务器端编程都使用 C# 语言进行。

第 2 部分为使用 ASP.NET 进行 Web 应用开发的核心内容，包括：

第 4 章　ASP.NET 基本控件，介绍了 ASP.NET 服务器控件

的基本概念，还介绍了 Label、Button、TextBox、HyperLink、CheckBox、RadioButton、ListBox、DropDownList、Panel、Image 和 ImageMap 等基本控件的使用方法。

第 5 章　ASP. NET 高级控件，在第 4 章的基础之上，进一步介绍了一些功能更完整、更有针对性的控件，包括 Calendar、FileUpload、Wizard、PlaceHolder、AdRotator 和验证等控件。

第 6 章　构建网站，介绍了 ASP. NET 网站的组织及控制机制，并重点介绍了 Response、Request、Application、Session 和 Server 对象的使用方法。

第 7 章　应用 ADO. NET 编程。ADO. NET 为 ASP. NET 提供高效的数据访问机制，本章介绍了使用 ADO. NET 对象访问数据库的基本方法。

第 8 章　Web 数据访问，介绍了通过 ASP. NET 提供的 Web 数据控件对数据库进行访问的方法，重点介绍了 GridView、DataList 和 DetailsView 控件的使用方法。

第 9 章　数据绑定，介绍了将 ADO. NET 数据库访问所得到的结果集作为数据源绑定到 Web 数据控件上的方法。

第 10 章　高级网站技术，介绍了在实用网站建设中非常重要的一些高级网站技术，包括母板页、导航、用户控件和网站的部署等内容。

第 11 章　实用编程技巧，介绍了一些 Web 应用开发中的实用技巧。该章每一节都可独立成篇，每一节的学习都可以丰富读者的开发技能与技巧。

第 12 章　高级数据库技术，介绍了一些高层次的数据库操纵技术。了解这些技术，有利于从更高的层次上完成系统设计，有利于实现更复杂的业务逻辑。

第 3 部分给出了一个综合应用实例，包括：

第 13 章　畅想网络学院，综合运用前面所学的各项技术，完整地实现了一个网上教学与管理的平台——“畅想网络学院”系统。

在本书的编写过程中，得到了很多同志的帮助，作者在此深表谢意。陈鸣教授对本书的编写提出了指导性意见，作者在此特别表示感谢。还要感谢王琦讲师和徐正芹讲师，他们首先在教学中采用了本书的内容，并对本书的修改提出了宝贵的建议。

由于作者水平有限，书中难免会存在一些不足之处，敬请读者批评指正。作者的电子邮箱是：njgaoyi@yahoo. com. cn. 。

目录 Contents

第 1 章 chapter 1

Web 应用基础

有些编程工具被描绘成不需要任何背景知识,不需要学习,就能完成专业效果的系统开发工作,到现在为止,我们很难同意这种说法。

软件开发是一项专业性很强的工作。有的开发工具可能很容易上手,但要真正用它来完成实际系统的开发,总有许多细节需要学习。当然,这些学习有的可以在实际开发工作中进行。另外,任何实际应用系统的开发都会涉及很多背景知识,这些背景知识有些是技术的,有些甚至是计算机以外的。

Microsoft Visual Studio 2005(VS2005)就是当前最高效的开发工具之一,它的集成开发环境(Integrated Develop Environment,IDE)为应用程序开发提供了极大的帮助。本书的主要内容就是介绍使用 VS2005 开发基于 Web 的网络应用程序所需要了解的一些重要细节。

要想真正地设计开发基于 Web 的网络应用程序,首先需要对与计算机网络相关的基本知识有所了解。另外,如果能对其他一些 Web 应用基础知识有所了解,如 HTML 标记语言、CSS 层叠样式表和客户端脚本语言等,在设计和开发中会更加得心应手。

上述每一方面的 Web 应用基础知识均包括很丰富的内容,都有专门的书籍进行介绍。在此,本章仅对 Web 应用程序设计开发中所不可回避的几方面知识进行简单的介绍,每方面的知识被压缩成一节,因此也只能是概念性的介绍。如果读者没有相关的知识背景,学习了本章的内容也足够开始工作了。当然,要想了解更全面的内容和深入的细节,还需要参考相关的专用文档。如果读者已经有了这方面的知识基础,完全可以跳过本章相关的内容甚至是整章而不会影响后面章节的学习。

1.1 HTML 标记语言

1.1.1 HTML 的基本概念

万维网(World Wide Web,WWW)以客户/服务器(client/server,C/S)方式工作,浏览器就是客户程序,万维网文档所驻留的计算机称为 Web 服务器。客户程序向服务器程序发出请求,服务器程序向客户程序送回所请求的万维网文档。在一个客户程序主窗口上显示出的万维网文档称为页面(page)。

万维网使用统一资源定位符(uniform resource locator,URL)来标识万维网上的各种文档,并使每一个文档在整个因特网范围内具有唯一性。万维网客户程序与服务器程序之间的交互使用超文本传输协议(hyper text transfer protocol,HTTP)。HTTP 是一个应用层协议,它使用 TCP 连接进行可靠的传送。万维网文档的内容使用超文本标记语言(hyper text markup language,HTML)表示,便于客户程序(浏览器)进行统一的解析和展示。

HTML 文件是以 HTML 语言表示内容的文本文件,以 .htm 或 .html 为扩展名。HTML 文件适合表示静态内容,而万维网上需要表示的大量动态内容,根据应用服务器及所使用开发语言的不同,扩展名可能是 .asp、.aspx、.jsp 或 .php 等。它们有不同的方法来处理动态内容,但都必须以 HTML 语言为基础,因为最终在客户端都要转化成 HTML 后才能展示。

本节就先对 HTML 语言做一个简单介绍。

首先来看一个最简单的 HTML 文档。可以用任何文本编辑器(如记事本)输入下面内容。当然,如果使用 FrontPage、Dreamweaver 一类的专用编辑器,则可以利用其所见即所得的编辑效果,更方便地完成任务。

```
<html>
<head>
<title>Hello World!</title>
</head>
<body>
<h1>Hello World!</h1>
This is my first page.
</body>
</html>
```

将文件保存为 HelloWorld.htm。这已经是一个真正的页面了,在操作系统下双击该文件,即可用默认的浏览器将其打开,界面如图 1-1 所示。



图 1-1　一个最简单的页面

下面对此 HTML 文档的内容做一个简单解释:

- 在 HTML 文档中,第一个标签是<html>。这个标签告诉浏览器这是 HTML 文档的开始。HTML 文档的最后一个标签是</html>,这个标签告诉浏览器这

是 HTML 文档的结束。

- 在＜head＞和＜/head＞标签之间的文本是头信息。在浏览器窗口中，头信息是不被显示的。
- 在＜title＞和＜/title＞标签之间的文本是文档标题，它被显示在浏览器窗口的标题栏。
- 在＜body＞和＜/body＞标签之间的文本是文档的正文部分，会被显示在浏览器中。

可见，HTML 文档的主要内容包含在各类标签中。下面介绍 HTML 文档中的几类重要标签。

1.1.2　HTML 元素

万维网文档之所以被称为 HTML 文档，是因为其内容都是由 HTML 标签进行标记的。

HTML 标签用来组成 HTML 元素，是一些由两个尖括号“＜”和“＞”括起来的预定义标记。HTML 标签通常成对出现，如＜h1＞和＜/h1＞，分别称为开始标签和结束标签。结束标签一般是在开始标签的标识符前面加一个斜杠，开始标签和结束标签再加上它们之间所包含的内容，就构成了一个 HTML 元素。HTML 标签与大小写无关。

HTML 标签可以带有属性，如＜h1 align＝"center"＞，表示该标题居中显示。属性定义通常是属性名和值成对出现，其形式为 name＝"value"。属性值应该被包含在引号中。双引号是最常用的，但是单引号也可以使用。在很少情况下，例如属性值本身包含双引号，使用单引号就很有必要了，如：

```
text='清除"标题"的内容'
```

HTML 元素有很多种，下面介绍其中最常用的几类。

1. 标题和段落

HTML 文档的文本内容主要由标题和段落元素表示。

标题元素用标签＜h1＞～＜h6＞定义，其中＜h1＞最大，＜h6＞最小。如果文档中有下面的内容：

```
<h1>这是一个标题</h1>
<h2>这是一个标题</h2>
<h3>这是一个标题</h3>
<h4>这是一个标题</h4>
<h5>这是一个标题</h5>
<h6>这是一个标题</h6>
```

则其页面效果如图 1-2 所示。

这是一个标题
这是一个标题
这是一个标题
这是一个标题
这是一个标题
这是一个标题

图 1-2　标题元素的显示效果

文档的段落用＜p＞标签定义，如：

```
<p>这是一段。</p>
```

当需要结束一行，但又不想开始新段落时，使用
标签。

```
<p>这又是一段,<br>
但有一个换行。</p>
```


标签不需要结束标签，它出现在哪里，就在哪里换行。很多页面设计者更倾向于不使用段落标签，而直接将内容放在<body>标签之间，用
强制换行来形成段落，如：

```
内容不一定非要在段落中,<br>
只要用 &lt;br&gt;换行就行了。
```

这是一段。

这又是一段，
但有一个换行。

内容不一定非要在段落中，
只要用
换行就行了。

图 1-3　文档的段落

上述内容的页面效果如图 1-3 所示。

可以看出，使用段落标签会在段落间形成较大的间隔，而
形成的换行则比较紧凑，往往更适合显示文本段落。

2. 超链

HTML 使用<a>标签来创建一个连接到其他文件的链接，称为超链（hyper link）。超链可以指向网络上的任何资源，包括 HTML 页面、声音、图像和视频等。

创建一个超链的语法为：

```
<a href="url">显示的超链名</a>
```

其中 href 属性用来指定连接到的地址，在开始标签<a>和结束标签中间的部分将作为超链显示在页面上。如：

```
<a href="http://www.microsoft.com/">微软</a>
```

在超链中可以使用 target 属性定义在哪里打开链接所指向的页面。如：

```
<a href="http://www.microsoft.com/" target="_blank">微软</a>
```

target 属性有如下枚举值可供选择：_blank、_parent、_search、_self 和_top，还可以是目标 frame 的名称。

3. HTML 框架

在页面中使用框架可以将页面窗口划分为不同的区域，在这些区域中可以同时显示多个 HTML 文档，其语法为：

```
<frameset>
    <frame src="url">
    <noframes>…</noframes>
</frameset>
```

frameset 元素中可以包含多个 frame，还可以再嵌套地包含内部 frameset 元素。在 frameset 元素中的 frame 和 frameset 只能以行或列的形式排列，并可分别定义它们在页面窗口中所占的大小。<noframes>标签中的文字将只出现在不支持 frameset 的浏览

器中。

下面这个例子中,将整个窗口分为上下两栏。上面的栏中显示 HTMLPage1.htm 页面,占 20%的高度;下面又是一个 frameset,占 80%的高度。下面的 frameset 又分为两栏,左边的栏中显示 HTMLPage2.htm 页面,占 25%的宽度;右边的栏中显示 HTMLPage3.htm 页面,占 75%的宽度。

```
<html>
<frameset rows="20% ,80% ">
    <frame src="HTMLPage1.htm">
    <frameset cols="25% ,75% ">
        <frame src="HTMLPage2.htm">
        <frame src="HTMLPage3.htm">
    </frameset>
</frameset>
</html>
```

页面效果如图 1-4 所示。

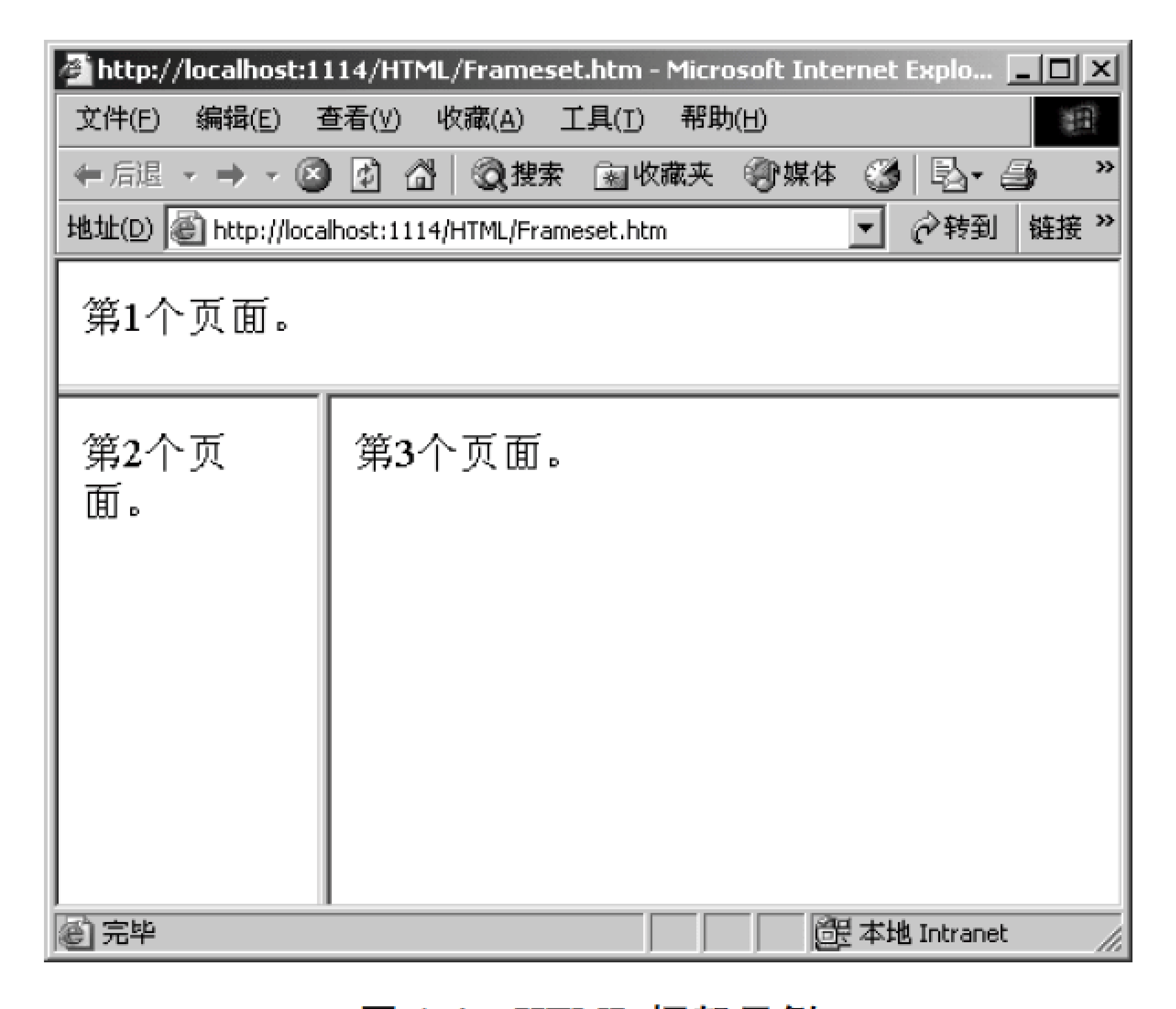


图 1-4 HTML 框架示例

frameset 和 frame 还可以指明 name、ID 和 noresize 等属性,frame 还可以作为超链的目标。

提示:如果浏览器支持 DHTML,可创建内嵌浮动框架 iframe,以达到更灵活的界面显示效果。

iframe 也是框架的一种形式,它与 frame 不同的是,可以嵌在网页中的任意位置。请看下面的例子:

```
<iframe left="100" top="80" width="480" height="320" src="URL">
</iframe>
```

本书对 iframe 不做详细介绍。

4. HTML 表单

用户与浏览器的交互是通过一些交互控件(如文本框、密码框、下拉列表、单选框和复选框等)来完成的。这些交互控件必须放在表单元素中才能通过表单元素最终完成与服务器的交互。下面是一个使用表单的例子:

```
<html>
<body>
    <form method="get" action="HTMLPage2.htm" target="_blank">
        用户名:<input id="Text1" type="text"/><br>
        口  令:<input id="Password1" type="password"/><br>
        类  型:<select id="Select1">
            <option value="S">学生</option>
            <option value="T">教师</option>
            <option value="M" selected>管理人员</option>
        </select><br>
        <input id="Submit1" type="submit" value="登录"/>
    </form>
</body>
</html>
```

页面执行效果如图 1-5 所示。

图 1-5 HTML 表单示例

在 form 标签中有如下属性。

- method 属性:用来指明通过 HTTP 协议提交表单数据的方式,有两个可能的选择:post 和 get。get 是默认方式,但 post 更为常用,因为使用 get 方式只能提交不超过 128 个字节的数据。
- action 属性:用来指明服务器端代理,当表单数据被提交到服务器端时,该代理会处理表单数据。示例中的代理是另一个页面,还可以是 CGI 程序、服务器端脚本和 Java Servlet 对象等。
- target 属性:用来指明响应页面在哪里打开。

下面对示例中涉及的几种交互控件做一个简单说明,这几种交互控件也是表单中最常用的。

文本框用于接受用户输入的文本并显示,如:

```
<input id="Text1" type="text"/>
```

用户输入的文本可以是一行,也可以是多行。

密码框用于输入密码,与文本框的区别是用户输入不回显,如:

```
<input id="Password1" type="password"/>
```

提交按钮:

```
<input id="Submit1" type="submit" value="登录"/>
```

当用户单击提交按钮时，表单的内容会被提交到服务器，由 form 的 action 属性所指明的代理进行处理。

上述三种交互控件都由<input>标签进行定义，根据 type 属性加以区分。

以下示例为一个简单的下拉式列表：

```
<select id="Select1">
    <option value="S">学生</option>
    <option value="T">教师</option>
    <option value="M" selected>管理人员</option>
</select>
```

下拉式列表控件由<select>标签进行定义，其中可包含多个选项，每个选项由<option>标签进行定义。初始选中的选项需要设置 selected 属性。

5. HTML 表格

HTML 表格由<table>标签定义。每个表格被划分为多个行(使用<tr>标签)，每行又被划分为多个数据单元格(使用<td>标签)。文档内容放在 td 元素中，这些内容可以是文本、图像和表单等几乎所有的 HTML 元素，还可以是嵌套的表格。表格是 HTML 文档中应用非常多的一个元素，除了显示需要表格化的内容外，还经常用它来规划窗口的布局。下面是一个简单的表格示例：

```
<html>
<body>
    <table width="80% ">
        <tr>
            <td>
                第 1行,第 1列</td>
            <td>
                第 1行,第 2列</td>
            <td>
                第 1行,第 3列</td>
        </tr>
        <tr>
            <td>
                第 2行,第 1列</td>
            <td colspan="2">
                合并的单元格</td>
        </tr>
    </table>
</body>
</html>
```

页面执行效果如图 1-6 所示。

用表格规划窗口布局与使用框架不同。使用框架需要将内容放在多个 HTML 文档中；而使用表格，所有的内容都在同一个文档中。因此，当内容需要变化时，一般使用框架；如果不需要变化，一般使用表格。

第1行，第1列	第1行，第2列	第1行，第3列
第2行，第1列	合并的单元格	

图 1-6　HTML 表格示例

6. HTML 图像

使用 img 标签可以在 HTML 文档中显示图像，如：

```
<img src="title0.jpg"/>
```

在 HTML 文档中还可以使用图像作为页面或表格等元素的背景，如：

```
<body background="39.jpg">
```

还可以用图像作为超链，如：

```
<a href="http://www.ilc.net.cn">
    <img border="0" src="ilc.gif">
</a>
```

随着 Web 应用的发展，用户对网页界面的要求越来越高，网页对图像的处理功能也随之越来越强，本书将在后面的章节中进一步介绍。

7. HTML 中的注释

使用注释标签可以在 HTML 文档中插入注释，如：

```
<!--注释内容 -->
```

注释会被浏览器忽略。使用注释对复杂代码加以说明是一个好的编程习惯。

1.2　CSS层叠样式表

1.2.1　什么是 CSS

CSS(Cascading Style Sheet，层叠样式表)也叫级联样式表，是由国际标准组织机构(World Wide Web Consortium，W3C)提出的，为了弥补 HTML 在排版样式上的不足而制定的一套样式标准。CSS 扩充了 HTML 各个标签的属性设置，使网页的视觉效果有更多变化。在当前的网页设计上，虽然基本语法还是 HTML，但样式的设定则更倾向于使用 CSS 来取代 HTML 的标签属性。也就是说，Web 页面的内容仍由 HTML 表示，但页面上各个元素的表现和布局则由 CSS 来控制。

CSS 除了重新定义了 HTML 原有的样式外(如文字的大小、颜色等)，还加入了重叠文字、层变化及任意位置的摆放等，使网页的编排与设计更具有灵活性，CSS 延伸了

HTML的功能。

当前所有主流浏览器都支持样式表。CSS有1.0版及2.0版，其中Netscape及IE的4.0以后版本均支持CSS 1.0，而IE从6.0版开始完全支持CSS 2.0。

简单地说，CSS可以做如下三件事：

① 设置页面中的字体。

② 定义层及其在页面中的位置。

③ 修改HTML标记。

使用CSS可以得到以下好处：

(1) 减少图形文件的使用。提供许多文字样式的设定，加上IE5、IE6内建的滤镜特效，取代原本需要图形才能呈现的视觉效果，减少了网页因为大量使用图形导致的下载速度变慢问题。

(2) 集中管理样式内容。以往在HTML文件中的样式设定分散在各个HTML标签内。CSS将网页的“样式”与“内容”的设定分开，也就是将网页的样式设定独立出来，便于对网页外观的统一控制与修改，便于保持网站风格的一致性。

(3) 共享样式设定。CSS样式设定的信息可以保存为独立的文件，让多个网页文件共同使用。因此，可以不必在每一个网页中做相同的样式设定了，也便于今后的统一修改。

(4) 将样式分类使用。同一个样式设定可以提供给不同的HTML文件使用，多个样式经过分类后也可以提供给一个HTML文件使用。

CSS使用规则(Rule)的方式进行定义，如：

```
h1 {color: green}
```

如果定义了上面的规则，则所有由<h1></h1>标签包含的文本都将以绿色显示。一条规则包含如下两个部分。

- 选择器(selector)：表示样式要套用的对象，如h1。
- 样式定义(declaration)：设定样式的内容，用{}括起来的部分，如{color: green}。

样式定义中可以包含多个属性的定义，每个属性的定义包括两部分：

```
属性：值
```

如：

```
color: green
```

多个属性定义之间用分号隔开。

1.2.2 CSS的使用

在Web页面中使用CSS有以下4种方法：

(1) 在HTML标签中使用行内(in-line)样式。

(2) 在HTML页面中嵌入(embed)一个样式表。

(3) 在HTML页面中链接(link)一个外部样式表文件(.css)。

(4) 通过 import 关键字导入(importing)样式表。

下面对这 4 种方法分别做一个简单介绍。

1. 在 HTML 标签中使用行内样式

若只需要修改 HTML 中某个标签的样式,可以使用标签的 style 属性直接修改样式。它利用现有的 HTML 标记,把特殊的样式加入到指定的标签中,又称行内设定,如:

```
<p style="color: red">这里显示红色字</p>
```

这种设定只对当前标签起作用,不影响其他相同类型的标签。

2. 在 HTML 页面中嵌入一个样式表

可以直接在 HTML 网页的标题部分(<head>和</head>标签之间),使用<style></style>标签来定义样式,用此方法定义的样式将应用于整个文档。例如:

```
<style>p {color: red;font-weight: bold}</style>
```

这样会对页面中所有<p>标记都起作用。

3. 在 HTML 页面中链接一个外部样式表文件

一个外部定义的样式表可以作为一个样式模板应用于多个 Web 页面。因此,如果有许多网页需要有相同的外观设计时(经常会有这种情况),则可将样式部分独立出来形成一个 CSS 文件。这样,这些网页就可以用外部链接的方式套用这些样式了,方法如下:

将所有样式部分组合在一起,保存到扩展名为 .css 的文件。因为<style>标签是 HTML 的一部分,因此在 .css 文件中就不必使用<style>标签了。

在 HTML 网页的<head>和</head>标签之间加上:

```
<link rel="stylesheet" type="text/css" href="样式文件(.css)的路径"/>
```

如:

```
<link rel="stylesheet" type="text/css" href="..\common\StyleSheet.css"/>
```

如果有多个样式文件要使用,只要使用多个<link>标签链接即可。

4. 通过 import 关键字导入样式表

使用@import 关键词从外部导入样式表,@import 在浏览器读取 HTML 之初便将样式文件的内容全部导入 HTML 文件中。

导入的方式是将样式文件的内容添加到 HTML 文档中,因此,@import 必须放在<style></style>标签之间。可见,通过 import 关键字导入的方式实质上就是在 HTML 页面中嵌入样式表的方式。

1.2.3 选择器

CSS 通过选择器(selector)来指定样式所要套用的对象,如 h1。但如果不希望整个

网站中同样的标签都使用相同的样式，也不想在每个标签上单独调整，则可以使用样式选择器做弹性改变。样式选择器有多种形式，这里仅介绍最常用的：类选择器(Class)、ID 选择器(ID)和上下文选择器(Contextual)。

1. 类选择器

使用类选择器可以在不同的标签上套用相同的样式。其定义语法为：

```
.类名称 {规则 1; 规则 2; …}
```

如：

```
.text1 {font-size: 12px; color: blue;}
```

在 HTML 中使用类选择器的语法为：

```
<标签名称 class="类名称">
```

如：

```
<table border="0" class="text1">
```

注意：定义时类名称前必须有一个点(.)，而使用时则不需要。

2. ID 选择器

ID 选择器和类选择器很相似，也是用来区别套用的样式。ID 选择器的定义语法为：

```
#ID标识符 {规则 1; 规则 2; …}
```

在 HTML 中使用 ID 选择器的语法为：

```
<标签名称 ID="ID标识符">
```

3. 上下文选择器

为 HTML 中的特定上下文定制的样式，其定义语法为：

```
标签 1 标签 2 {规则 1; 规则 2; …}
```

在上下文选择器中，标签名称之间用空格来分隔，表示在 HTML 中若有＜标签 2＞包含在＜标签 1＞当中时，即＜标签 1＞…＜标签 2＞…＜/标签 2＞…＜/标签 1＞，则套用样式。

注意：若定义时在标签 1 与标签 2 间用逗号(,)分隔，则表示对标签 1 及标签 2 分别套用这个样式。

1.2.4 CSS 文件样例

本书应用实例所使用的层叠样式表文件(～\common\StyleSheet.css)的内容为：

```
body
```

```
{
    font-size: 14px;
    font-family: 宋体,'Times New Roman';
    color: black;
}

//主页框架 table的属性,主要是为了加上虚线边框
//只在 mainpage.aspx 中使用
.all
{
    border-bottom-width: 1px;
    border-bottom-style: dotted;
    border-bottom-color: gray;
    border-top-width: 1px;
    border-top-style: dotted;
    border-top-color: gray;
    border-right-width: 1px;
    border-right-style: dotted;
    border-right-color: gray;
    border-left-width: 1px;
    border-left-style: dotted;
    border-left-color: gray;
}
.right
{
    border-left-style: none;
    border-right-width: 1px;
    border-right-style: dotted;
    border-right-color: gray;
}

//从 Office 转换过来的文档字体默认属性
.MsoNormal
{
    font-size: 12px;
    color: gray;
}

//各功能页面的标题属性
.title
{
    font-size: 22pt;
    color: green;
    font-family: 华文新魏;
```

```
    text-align: center;
}

//灰色的小字
.text
{
    font-size: 12px;
    color: gray;
}

//蓝色的小字
.text1
{
    font-size: 12px;
    color: blue;
}

//默认字体
.text2
{
    font-size: 14px;
    font-family: 宋体,'Times New Roman';
    color: black;
}
```

上述CSS文件中先定义了HTML文档中body部分的默认属性,又定义了几个样式类。文件中有相应的注释,这里不再详细解释。

1.3 JavaScript语言

1.3.1 JavaScript语言概况

使用HTML语言和CSS技术已经可以制作漂亮的页面了,但这样的页面仍然存在一定的缺陷,如页面的内容为静态内容;缺少用户与客户端浏览器的动态交互。

为了解决这一问题,出现了多种客户端的脚本语言。它们可以使得浏览器和用户之间不仅是一种显示和浏览的关系,而且实现了一种实时的、动态的、交互式的表达能力。

在所有的脚本语言中,最常用的有JavaScript、VBScript等,本节对JavaScript做一个简单介绍。

JavaScript语言与C语言和Java语言在很多方面非常相似,因此在说明时会与这两种语言进行参照。我们认为读者已经具有C语言或者Java语言的基础,因此,对于JavaScript中与C语言和Java语言相同的部分会介绍得简略一些。

JavaScript是一种轻型的、解释型的程序设计语言,并且还是一种基于对象(object)

的、事件驱动(event driven)的、安全的脚本语言。使用它的目的是为 Web 页面增加客户端的交互功能。可以通过嵌入或引入的方式将 JavaScript 脚本加入到标准的 HTML 语言中。JavaScript 是 Java 与 HTML 折中的选择,具有以下基本特点。

(1) 一种脚本编程语言。JavaScript 是一种脚本语言,它采用小程序段的方式实现编程。像其他脚本语言一样,JavaScript 同样也是一种解释性语言。

(2) 基于对象的语言。JavaScript 是一种基于对象的语言,在 JavaScript 中可以使用预创建的对象。

(3) 简单性。JavaScript 类似于 Java,但比 Java 简单。它是一种弱类型的语言,并未使用严格的数据类型,变量也不必先声明再引用。

(4) 安全性。JavaScript 是一种安全的语言,它不允许访问本地的硬盘,不能将数据存到服务器上,不允许对网络文档进行修改和删除,只能通过浏览器实现信息浏览或动态交互,从而有效地增强了数据的安全性。

(5) 动态性。JavaScript 是动态的,它可以直接对用户的输入做出响应,无需经过 Web 服务器处理。它对用户输入的响应是以事件驱动的方式进行的,直接在客户端完成,不需要传回服务器进行处理,从而能够得到很快的响应效果。

(6) 跨平台性。JavaScript 依赖于浏览器,与操作环境无关。只要浏览器支持 JavaScript,它就能正确执行。

先来编写一个最简单的 JavaScript 程序。通过它可说明 JavaScript 的脚本是怎样被嵌入到 HTML 文档中的。

创建一个 HTML 页面 HelloWorld.htm,其代码为:

```
<html>
<head>
    <title>使用 JavaScript</title>
<script language="JavaScript" type="text/JavaScript">
    //这是第一个 JavaScript 程序
    alert("Hello World!");
    mess="欢迎使用 JavaScript!";
    alert(mess);
</script>
</head>
<body>
    使用 JavaScript。
</body>
</html>
```

执行此页面,浏览器会先弹出一个 Hello World!对话框,如图 1-7(a)所示。用户单击"确定"按钮后,会再弹出一个"欢迎使用 JavaScript!"对话框,如图 1-7(b)所示。用户再次单击"确定"按钮,浏览器才继续显示文档内容。

说明:同 HTML 标记语言一样,JavaScript 程序代码也可以用任何编辑软件进行编辑,被插入到 HTML 的相关区域。



(a)



(b)

图 1-7 第一个最简单的 JavaScript 程序

JavaScript 代码由<script language="JavaScript"></script>标签说明。在标签之间加入 JavaScript 脚本。

alert()是 JavaScript 的窗口对象方法,其功能是弹出一个具有"确定"按钮的对话框,并显示作为参数的字符串。

与 Java 语言相同,JavaScript 也支持两种注释方式:/ * … * /型和//型。

JavaScript 是一种区分外文字母大小写的语言。

JavaScript 中标识符的定义也与 Java 相同:第一个字符必须是字母、下划线(_)或美元符号($),接下来的字符可以是字母、数字、下划线和美元符号。

1.3.2 JavaScript 基本数据类型

JavaScript 支持三种基本数据类型:数字、文本字符串和布尔型。此外,还支持两种特殊数据类型 null(空)和 undefined(未定义),它们各自只定义了一个值。除了基本数据类型外,JavaScript 还支持复合数据类型:对象和数组。

在 JavaScript 的基本类型中,数据可以是常量,也可以是变量。JavaScript 是弱类型的,一个数据的变量或常量不必事先声明,而是可以等到使用时再确定其数据类型(当然也可以先声明后引用)。

1. 常量

JavaScript 的常量通常又称字面常量,是其值不能改变的数据。

- 整型常量:可以使用十六进制、八进制和十进制表示其值。
- 实型常量:由整数部分加小数部分组成,如 12.32、193.98 等。也可以使用科学表示法,如 5E7、4.3e5 等。
- 布尔常量:只有 True 和 False 两个值。
- 字符型常量:使用单引号(')或双引号(")括起来的多个(可以是 0 个)字符。如"Hello World!"、"欢迎使用 JavaScript!"和'China'等。同 C 语言一样,JavaScript 中同样可以使用反斜杠(\)来进行字符的转义。
- 空值:null,表示什么也没有。如试图引用没有定义的变量,则返回一个 null 值。
- 未定义:undefined。这个值并不常用,当使用了一个并未声明的变量,或使用了已经声明但还没有赋值的变量,或者使用了一个并不存在的对象属性时,返回的就是这个值。

2. 变量

JavaScript 语言与 C 语言和 Java 语言的一个重要区别就是：JavaScript 是弱类型的，也就是说 JavaScript 的变量可以存放任何类型的值。在 JavaScript 中，可以先把一个数值赋给一个变量，然后再把一个字符串赋给它。如下面的语句是合法的：

```
i=20;
i="Hello World!";
```

在 JavaScript 中，变量可以用命令 var 声明并初始化，如：

```
var i;
var mess,str="China";
```

在 JavaScript 中同样区分全局变量和局部变量。全局变量定义在所有函数体之外，其作用范围是这个变量定义之后的所有语句；而局部变量在函数体内定义，只在定义它的函数中可见。

1.3.3 函数与事件驱动

1. 函数

在 HelloWorld.htm 中，JavaScript 语句被直接包含在<Script></Script>标签之间，这些语句在页面加载时执行。

JavaScript 的功能还可以由函数来完成，这些功能在函数被调用时才执行。

JavaScript 函数由关键字 function 定义。函数定义的语法为：

```
function 函数名 (参数表)
{
    函数体;
    return 表达式;
}
```

函数定义之后，就可以通过"函数名(实参表)"的形式来对它进行调用。需要注意的是，函数名对大小写是敏感的。

在函数体中，表达式、运算符和控制语句的使用与 C 和 Java 非常相似。

JavaScript 的程序控制语句同样包括 if 条件语句、for 循环语句、while 循环语句、break 语句和 continue 语句等，这部分内容本书不做详述，请读者参阅相关书籍。

2. 事件驱动及事件处理

JavaScript 采用事件驱动的机制对用户输入做出响应。在图形界面环境下，通常称鼠标或键盘的动作为事件(event)。在 JavaScript 中，对事件的处理通常由函数完成，称为事件处理函数。表 1-1 为 JavaScript 所能处理的主要事件。

表 1-1 JavaScript 所能处理的主要事件

事件名	触 发 时 机
onClick	单击事件：当用户单击鼠标按钮时发生
onChange	内容改变事件：当利用 text 或 textarea 等控件所输入的值发生改变时引发该事件。另外，当在 select 列表中的选中项改变后也会引发该事件
onSelect	选中事件：当 text 或 textarea 等控件中的文字被加亮后，引发该事件
onFocus	获得焦点事件：当用户单击 text 或 textarea，以及 select 控件时，产生该事件。此时该控件成为当前输入焦点对象
onBlur	失去焦点：当 text 或 textarea，以及 select 控件不再拥有输入焦点而退到后台时引发该事件，它与 onFocus 事件是对应的关系
onLoad	载入文件：当文档载入时引发该事件。onLoad 的一个作用就是在首次载入一个文档时检测 cookie 的值，并用一个变量为其赋值，使它可以被源代码使用
onUnload	卸载文件：当 Web 页面退出时引发该事件，并可更新 cookie 的状态

下面是一个简单的页面实例，从中可以看出 JavaScript 事件的触发和处理机制。

```
<html>
<head>
    <title>使用函数</title>
<Script Language="JavaScript">
function Close()
{
    window.close();
}
</Script>
</head>
<body>
    <input id="Button1" type="button" value="关闭" onclick="Close()" />
</body>
</html>
```

在上述代码中，为“关闭”按钮指定了 onclick 事件的处理函数为 Close()。Close()函数的实现中只包含一句 window.close()，它调用 window 对象的 close()方法，关闭当前浏览器窗口。

注意：如果使用的浏览器是 IE 7.0，close()方法则是关闭当前选项卡。

3. 基于对象的 JavaScript 语言

对象是 JavaScript 中一类最重要的数据类型。

JavaScript 语言是基于对象的(object-based)，而不是面向对象的(object-oriented)。JavaScript 具有一些面向对象的基本特征，例如，它可以根据需要创建自己的对象，而且它还有自己的以原型对象为基础的继承机制。但 JavaScript 没有正式的类的概念，没有提供以类为基础的抽象、继承和重载等面向对象语言的典型功能。另外，JavaScript 是弱

类型的,这也不符合经典面向对象语言的要求。

JavaScript 中的对象是由属性(properties)和方法(methods)构成。

一个对象在被引用之前,这个对象必须存在,否则系统会给出出错信息。JavaScript 在引用对象之前,要么创建新的对象,要么利用现存的对象,即引用 JavaScript 内部对象或引用由浏览器环境提供的对象。

在 JavaScript 中提供了几个特殊的用于操作对象的语句、关键词及运算符。

(1) for…in 语句

格式如下:

```
for (变量 in 对象)
{
    执行语句
}
```

说明:该语句的功能是用于对已知对象的所有属性进行循环操作。它是将一个已知对象的所有属性反复置给一个变量,而不是使用计数器来实现。

该语句的优点就是无需知道对象中属性的个数即可进行循环操作。

(2) with 语句

该语句对一个对象起作用,在该语句体内引用该对象的属性,可以不用指明对象,直接使用属性。即在该语句体内,任何对变量的引用都被认为是对这个对象属性的引用,这样可以节省一些代码。其格式如下:

```
with object
{
    …
}
```

(3) this 关键词

this 是对当前对象的引用。在 JavaScript 中,由于对象的引用是多层次、多方位的,往往在一个对象的引用中又需要对另一个对象进行引用,为此 JavaScript 提供了 this 关键字用于指向当前对象。

(4) new 运算符

使用 new 运算符可以创建一个新的对象,在这个运算符之后必须有用于初始化对象的构造函数名,格式如下:

```
newObject=new object(实参表);
```

其中 newObject 是要创建的新对象,object 是已经存在的对象。

如创建新的日期和时间对象:

```
now=new Date();
birthday=new Date(1997,3,21);
```

JavaScript 为编程人员提供了一些非常有用的系统对象和系统函数。用户不需要自编脚本来实现这些功能,这就为编程人员快速开发功能强大的程序提供了便利条件。

JavaScript 提供的系统对象包括 String(字符串)、Math(数值计算)和 Date(日期)等。

(5) String 对象

String 对象可用于处理或格式化文本字符串,以及确定和定位字符串中的子字符串。String 对象最常用的属性是 length,它指出 String 对象中的字符个数,如:

```
myname="Michael Jordan"
mynamelength=myname.length
```

mynamelength 返回 myname 字符串的长度为 14。

String 对象有多个方法,主要用于字符串在 Web 页面中的显示、字体大小、字体颜色、字符的搜索及字符的大小写转换等操作。

① anchor()方法

该方法在对象字符串的两端放置一个具有 Name 属性的 HTML 锚点,其调用格式为:

```
strVariable.anchor(anchorString)
```

如:

```
var str="This is an anchor";
str=str.anchor("Anchor1");
```

上述代码执行后,str 的值为:

```
<A NAME="Anchor1">This is an anchor</A>
```

② 与字符显示有关的方法

- big():把 HTML 的<BIG>标记放置在 String 对象中的文本两端,使文本以大字体显示,如:

  ```
  var str="This is a String";
  str=str.big();
  ```

  上述代码执行后,str 的值为:

  ```
  <BIG>This is a String</BIG>
  ```

- italics():把 HTML 的<I>标记放置在 String 对象中的文本两端,使文本以斜体显示。
- bold():把 HTML 的<B>标记放置在 String 对象中的文本两端,使文本以粗体字显示。
- blink():把 HTML 的<BLINK>标记放置在 String 对象中的文本两端,使文本闪烁显示。

- small()：把 HTML 的<SMALL>标记放置在 String 对象中的文本两端，使文本以小体字显示。
- fixed()：把 HTML 的<TT>标记放置在 String 对象中的文本两端，使文本以固定高亮字显示。
- fontsize(size)：把 HTML 的<FONT>标记放置在 String 对象中的文本两端，控制字体大小，如：

```
var str="This is a String";
str=str.fontsize(12);
```

上述代码执行后，str 的值为：

```
<FONT SIZE="12">This is a String</FONT>
```

- fontcolor(color)：把 HTML 的<FONT>标记放置在 String 对象中的文本两端，控制字体颜色。

③ 字符串大小写转换

- toLowerCase()：返回一个字符串，将 String 对象中的字母转换为小写。
- toUpperCase()：返回一个字符串，将 String 对象中的字母转换为大写。

如：

```
var str="This is a String";
str=str. toLowerCase();
```

上述代码执行后，str 的值为：

```
this is a string
```

④ 字符搜索

- indexOf()：返回一个整数值，指出 String 对象内第一次出现子字符串的位置，其调用格式为：

```
strObj.indexOf(subString[,startIndex])
```

其中 strObj 为 String 对象；subString 为要在 String 对象中查找的子字符串，startIndex 为可选项。该整数值指出在 String 对象内开始查找的位置，如果省略，则从字符串的开始处查找。

位置计数从 0 开始，如果没有找到子字符串，则返回－1。

- substring()：返回 String 对象中指定位置的子字符串，其调用格式为：

```
strObj.substring(start,end)
```

其中 start 指明子字符串的起始索引，第 1 个字符的索引为 0；end 指明子字符串的结束位置，位置索引也是从 0 开始起算。如：

```
var str="This is a String";
str=str. substring(5,7);
```

上述代码执行后，str 的值为：

```
is
```

说明：substring()方法返回一个从 start 到 end(不包含 end)的子字符串。substring()方法使用 start 和 end 中的较小值作为子字符串的起始点，所以 str.substring(0,3)和 str.substring(3,0)将返回相同的子字符串。

- substr()：返回一个从指定位置开始的指定长度的子字符串，其调用格式为：

```
strObj.substr(start [,length ])
```

(6) Math 对象

Math 对象是一个固有对象，不能用 new 运算符创建，提供基本数学函数和常数。

Math 对象的属性主要用于返回一些常用的常数，包括 E 属性(自然对数的底，约等于 2.718)、LN2 属性(2 的自然对数，约等于 0.693)、LN10 属性(10 的自然对数，约等于 2.302)、LOG2E 属性(以 2 为底 e 的对数，约等于 1.442)、LOG10E 属性(以 10 为底 e 的对数，约等于 0.434)、PI 属性(圆的周长与其直径的比值，约等于 3.141592653589793)、SQRT1_2 属性(0.5 的平方根，约等于 0.707)和 SQRT2 属性(2 的平方根，约等于 1.414)等。

Math 对象提供多个方法，包括绝对值方法 abs()、正弦方法 sin()、余弦方法 cos()、反正弦方法 asin()、反余弦方法 acos()、正切方法 tan()、反正切方法 atan()、四舍五入方法 round()、平方根方法 sqrt()和幂方法 pow(base,exponent)等，其方法名大多具有自解释性，这里不再详述。

(7) Date 对象

Date 对象从系统取得日期和时间。

Date 对象保存以 ms 为单位的日期和时间，所能表示的日期范围大约是 1970 年 1 月 1 日前后的各 285616 年。

Date 对象是动态对象，即需要使用 new 运算符创建一个实例。如：

```
dateObj= new Date()
dateObj= new Date(dateVal)
dateObj= new Date(year,month,date[,hours[,minutes[,seconds[,ms]]]])
```

其中，如果 dateVal 是数字值，表示指定日期与 1970 年 1 月 1 日零时相差的毫秒数；如果是字符串，则按照 parse 函数中的规则进行解析。

注意：parse 函数解析一个包含日期的字符串，并返回该日期与 1970 年 1 月 1 日 0 时之间所间隔的毫秒数。

Date 对象没有提供可直接访问的属性，只具有获取和设置日期与时间的方法。

获取日期与时间的方法如下：

- getYear()方法返回年数。
- getMonth()方法返回月份数。

- getDate()方法返回日数。
- getDay()方法返回星期几。
- getHours()方法返回小时数。
- getMinutes()方法返回分钟数。
- getSeconds()方法返回秒数。
- getTime()方法返回毫秒数。

设置日期与时间的方法如下：

- setYear()方法设置年数。
- setDate()方法设置日数。
- setMonth()方法设置月份数。
- setHours()方法设置小时数。
- setMinutes()方法设置分钟数。
- setSeconds()方法设置秒数。
- setTime ()方法设置毫秒数。

(8) JavaScript 中的系统函数

JavaScript 中提供了多个系统函数，也称为内部方法。这些系统函数与任何对象无关，使用这些函数不需创建任何实例，可直接使用。其中最常用的系统函数包括：

① eval 函数

eval 函数允许 JavaScript 源代码的动态执行，其格式为：

```
eval(codeString)
```

其中 codeString 参数是包含有效 JavaScript 代码的字符串。这个字符串将由 JavaScript 分析器进行分析和执行，如：

```
eval('alert("Hello World!");');
```

上述代码执行的效果是：在弹出式对话框中显示“Hello World!”。相当于执行 alert("Hello World!");语句。

② escape 函数

escape 函数对 String 对象编码以便它们能在所有计算机上可读，其调用格式为：

```
str=escape(charString)
```

escape 函数返回一个包含了 charString 内容的 Unicode 格式字符串，其中所有空格、标点及其他非 ASCII 字符都用%xx 编码代替，其中 xx 为原字符的十六进制值。例如，空格返回的是"%20"。

③ unescape 函数

unescape 函数对用 escape 函数进行了编码的 String 对象进行解码，其调用格式为：

```
str=unescape(charString)
```

④ parseFloat 函数

parseFloat 函数返回从字符串转换来的浮点数值，其调用格式为：

```
floatVar=parseFloat(numString)
```

其中 numString 参数是包含浮点数的字符串。

parseFloat 函数返回 numString 所表示的数值。如果 numString 的前缀不能解释为浮点数，则返回 NaN。如：

```
parseFloat("abc")          // 返回 NaN
parseFloat("1.2abc")       //返回 1.2
```

注意：NaN 是表示算术表达式返回非数字值的一个特殊值。NaN 不与任何值相等，包括其本身。要检测值是否为 NaN，可使用 isNaN 函数。

⑤ parseInt 函数

parseInt 函数返回由字符串转换得到的整数，其调用格式为：

```
intVar=parseInt(numString,[radix])
```

其中 radix 参数可选，其值在 2～36 之间，为 numString 所表示数字的进制。如果没有提供该参数，则前缀为'0x'的字符串被当作十六进制数，前缀为'0'的字符串被当作八进制数，所有其他字符串都被当作是十进制数。

4. 内部对象

使用浏览器的内部对象系统，可在 JavaScript 中实现与 HTML 文档的交互。浏览器内部对象的作用是将 HTML 文档中的相关元素组织起来，提供给程序设计人员使用，从而减轻编程人员的劳动，提高 Web 页面的控制能力。

Web 浏览器的主要任务是在一个窗口中显示 HTML 文档。在客户端 JavaScript 中，表示 HTML 文档的是 Document 对象，Window 对象代表显示该文档的 Web 浏览器窗口或窗口中的一个框架。

Window 对象是 JavaScript 中的一个关键对象，其他所有的客户端对象都和 Window 对象有联系，都可以作为 Window 对象的属性来存取。例如，每个 Window 对象都包含一个 document 属性，该属性引用与这个窗口关联在一起的 Document 对象。此外，Window 对象还包含一个 frames[]数组，它引用代表原始窗口各子框架的 Window 对象。例如，window. document 代表的是当前窗口的 Document 对象，而 window. frames[0]. document 代表的是当前窗口的第一个子框架的 Document 对象。

JavaScript 的内部对象层次如图 1-8 所示。

下面对图 1-8 中所涉及的对象做一个简单说明。

- 浏览器对象（Navigator）：包含浏览器的总体信息，如版本等。
- 屏幕对象（Screen）：提供用户显示器的大小和可用的颜色数量等信息。属性 width 和 height 指的是以像素为单位的显示器大小。属性 availWidth 和

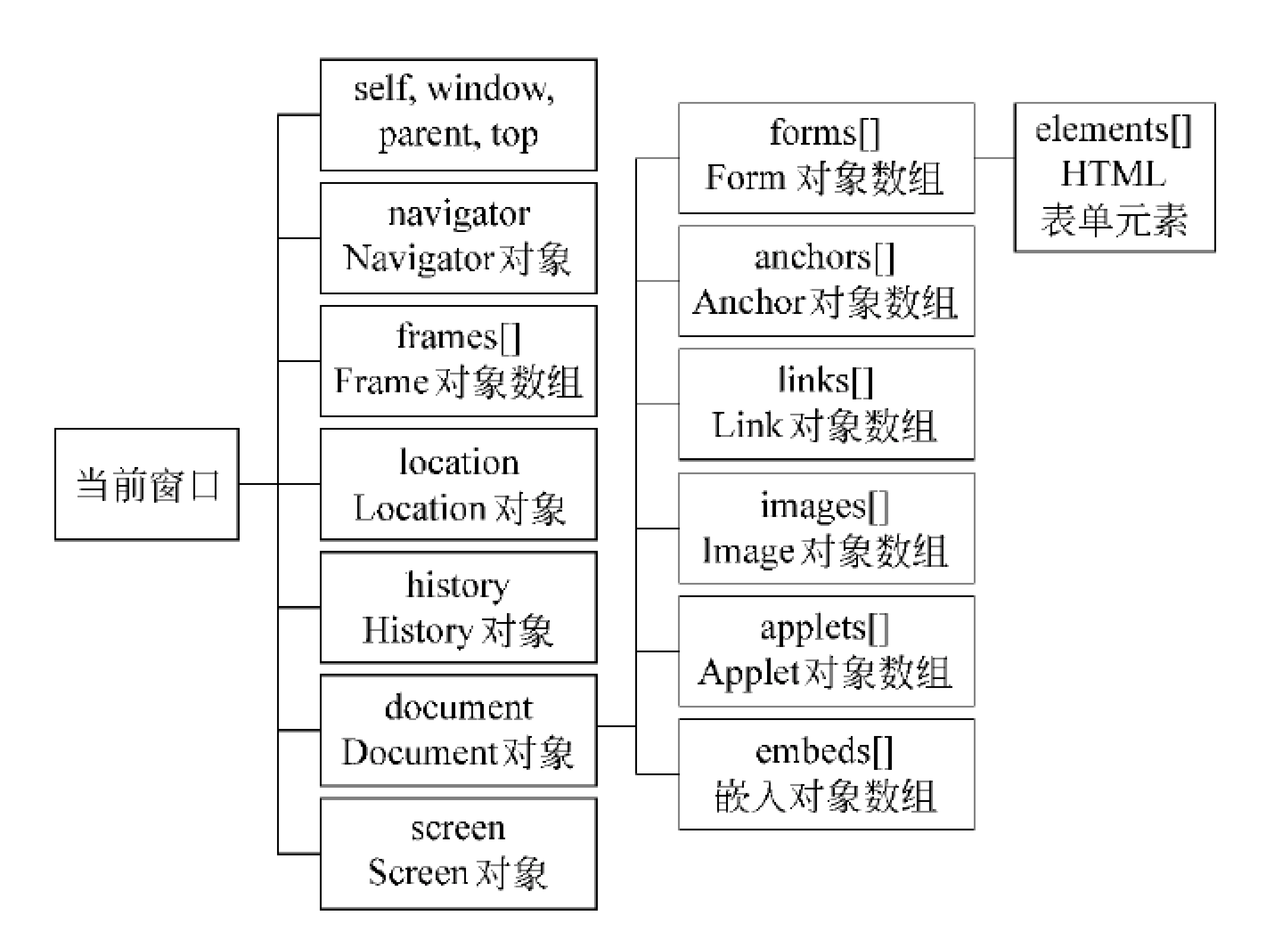


图 1-8　JavaScript 的内部对象层次图

availHeight 指的是实际可用的显示器大小。

- 位置对象(Location)：提供了当前打开文档的 URL。它的 href 属性是一个字符串，包含完整的 URL 文本。
- 历史对象(History)：提供了与访问历史有关的信息。
- 文档对象(Document)：每个 Window 对象都包含一个 document 属性，该属性引用一个 Document 对象，该 Document 对象表示显示在该窗口中的 HTML 文档。

Document 对象定义了以下 4 个关键方法。

- close()：关闭文档。
- open()：产生一个新文档，擦掉已有的文档内容。
- write()：把文本输出到当前打开的文档。
- writeln()：把文本输出到当前打开的文档，并附加一个换行符。

Document 对象定义了如下属性。

- anchor[]锚对象数组：代表文档中所有用标记的锚对象。
- links[]链接对象数组：代表文档中所有用标记的超文本链接对象。
- forms[]窗体对象数组：代表文档中所有用<form></form>标记的窗体对象。
- 其他属性：包括链接颜色属性 alinkColor、linkColor 和 vlinkColor，背景颜色 bgColor 及前景颜色 fgColor 等。

5. 窗口及输入输出

JavaScript 是基于对象的脚本编程语言，它的输入输出都是通过对象来完成的。其中输入可通过窗口对象的方法来完成，而输出可通过文档对象的方法来完成。下面是一个简单的例子：

```
<html>
```

```
<head>
    <title>JavaScript的输入输出</title>
<script language="JavaScript">
    var test=window.prompt("请输入任何内容：");
    document.write("刚才输入的内容为："+test);
</script>
</head>
<body>

</body>
</html>
```

其中：window.prompt()是窗口对象的一个方法，它的基本功能是当装入Web页面时在屏幕上显示一个具有“确定”和“取消”按钮的对话框时，让用户输入数据。document.write()是文档对象的一个方法，它的基本功能是实现向Web页面的输出显示。

习　　题

1. 使用文本编辑器打开一个HTML文档，观察文件的内容，并指出其中使用了哪些HTML元素。

2. 使用文本编辑器编写一个HTML文档，要求能够在浏览器中显示为如图1-9所示的形式。



图1-9　习题2图

3. 修改习题2中创建的HTML文档，分别实现1.1.2节中所介绍的段落、超链、框架、表单、表格、图像和注释等内容。

4. 什么是CSS层叠样式表？CSS在网络程序设计中有什么作用？

5. 在Web页面中使用CSS有哪些方法？

6. CSS是怎样通过选择器来指定页面样式的？

7. 参照1.2.2节所介绍的方法，依次在习题2中创建的HTML文档上使用CSS层叠样式表。

8. 在网页编程中引入JavaScript脚本语言有何意义？

9．JavaScript 语言有何特点？

10．JavaScript 语言支持哪些数据类型？

11．如何理解 JavaScript 语言是基于对象的，而不是面向对象的？

12．简述 JavaScript 都能处理哪些事件。

13．简述 for…in 语句的用法。

14．简述 JavaScript 语言中主要有哪些常用对象？

15．简述 Math 对象各属性的含义。

16．在习题 2 创建的 HTML 文档中增加一个名为“关闭”的按钮，并使用 JavaScript 语言为该按钮创建一个事件响应函数，当使用鼠标单击该按钮时，能够关闭当前页面。

17．JavaScript 都支持哪些浏览器内部对象？

18．什么是 Document 对象？简述其方法和属性。

19．简述 Window 对象与 Document 对象的关系。

第 2 章 chapter 2

ASP.NET 开发入门

本章介绍使用 ASP.NET 进行 Web 应用开发的最基本内容，包括 Microsoft Visual Studio 2005 的安装及其集成开发环境等。在这一章中将会建立第一个 Web 应用程序。

2.1 开发环境的建立

Microsoft Visual Studio 2005 是一个非常高效的开发工具，简称 VS2005。本书所介绍的基于 Web 应用程序的开发都是在 Visual Studio 2005 下完成的，所以本节先介绍 Visual Studio 2005 开发环境的建立过程。

VS2005 特别适合构建交互式 Web 站点，尤其是创建与服务器端数据库进行交互的动态 Web 站点。VS2005 在构建 Web 应用时使用的是 ASP.NET 2.0 技术。

ASP.NET 2.0 与之前的 ASP.NET 1.x 比较有很大的改进，使用 ASP.NET 2.0，只需要编写更少的代码就可以完成更强的功能。

2.1.1 安装 Visual Studio 2005

插入光盘自动执行或手工选择执行 setup.exe 程序，进入 Visual Studio 2005 安装的初始界面，如图 2-1 所示。

在图 2-1 所示的界面中单击“安装 Visual Studio 2005”，即可跟随 Visual Studio 2005 的安装向导逐步完成安装任务。

系统首先显示“安装程序正在加载安装组件”的进度条，加载完成后，系统显示“欢迎使用 Microsoft Visual Studio 2005 安装向导”界面，单击“下一步”按钮继续。

系统提醒 “请退出所有应用程序，然后再继续安装”，并列出将要安装的组件。选择“我接受许可协议中的条款”并输入产品密钥，输入用户名称，按“下一步”按钮继续。

在“选择要安装的功能”界面上选择“默认值”项(如果是有经验的用户，可以选择其他选项)，选择安装目录，单击“安装”按钮开始安装，如图 2-2 所示。

等待左侧列表中的组件全部安装完成后，如果系统提示“必须重新启动计算机才能完成安装”，请选择“立即重新启动”。系统重新启动后也许会出现一个等待提示，没关系，系统会自动继续安装并最终出现安装“成功”界面，单击“完成”按钮，返回到如图 2-1



图 2-1　Visual Studio 2005 安装的初始界面



图 2-2　Visual Studio 2005 的安装过程

所示的界面。

2.1.2　安装 MSDN Library

MSDN 是 Microsoft Software Developer Network 的简称，是 Microsoft 当前提供的有关编程信息的最全面的资源，包含上千兆字节的开发人员所必需的信息、文档示例代码、技术文章等，可供全世界的开发者使用。可以在 http://msdn.microsoft.com 看到有关软件开发的资料。

Visual Studio 2005 中包含 MSDN Library 的光盘,是针对 Visual Studio 2005 的最全面的联机帮助资料。如果出于硬盘空间的考虑,可以选择不安装 MSDN,但是在开发中就无法获得帮助,而在 MSDN 中包含了大量的帮助信息、文档和代码例程,对开发工作是大有帮助的。

当完成 Visual Studio 2005 的安装并再次回到如图 2-1 所示界面后,“安装产品文档”变亮,单击“安装产品文档”,可继续跟随向导安装 MSDN Library,如图 2-3 所示。

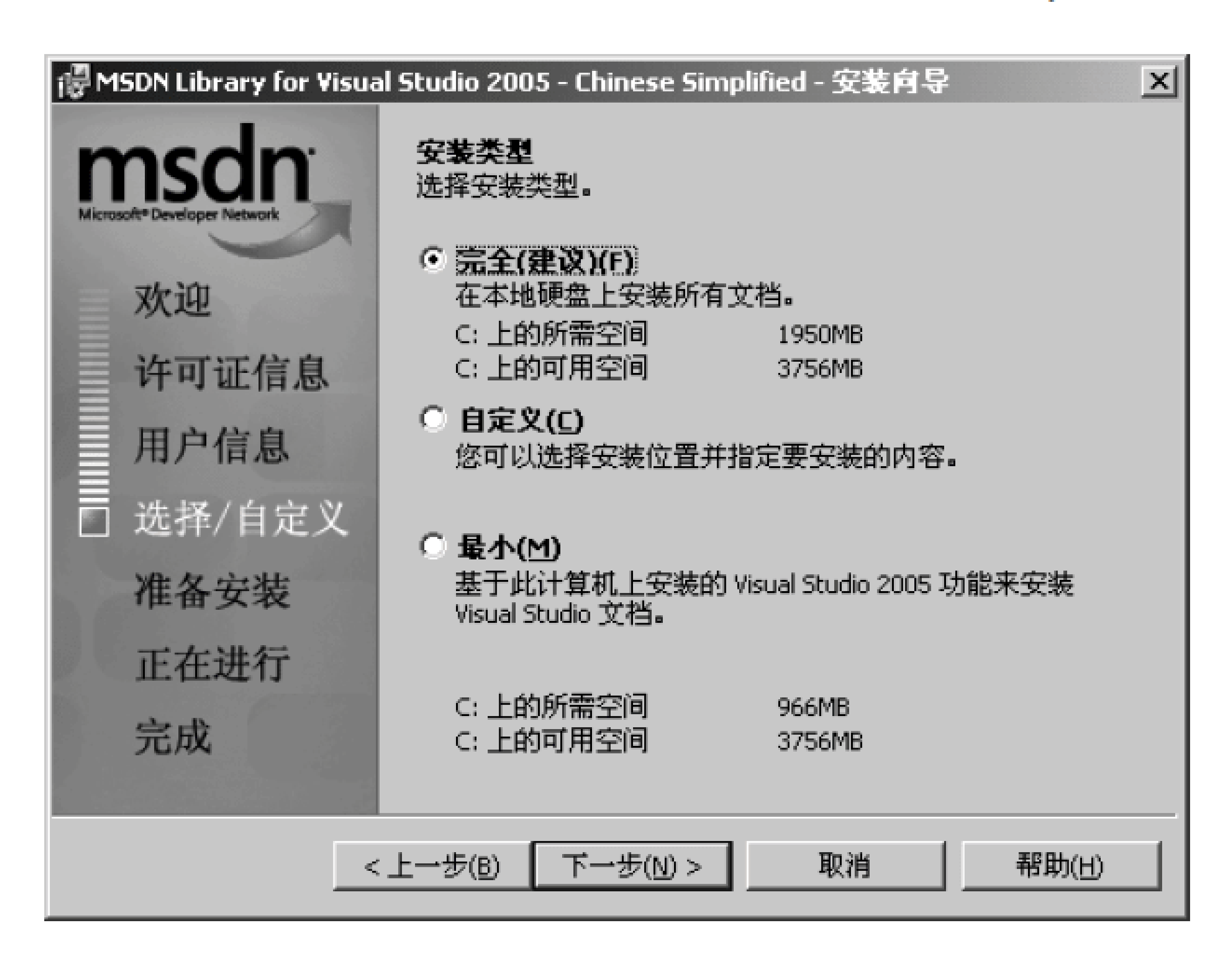


图 2-3 MSDN 的安装类型

首先是“欢迎”界面,单击“下一步”按钮继续。

在“许可协议”界面上选择“我接受许可协议中的条款”,单击“下一步”按钮继续。

在“客户信息”界面上输入用户名和单位,单击“下一步”按钮继续。

在“安装类型”界面上选择“完全”按钮安装,单击“下一步”按钮继续。

在“目标文件夹”界面上选择目标文件夹,单击“下一步”按钮继续。

在“准备安装程序”界面上单击“安装”按钮,系统开始复制文件。请耐心等待,直至出现“已完成安装”界面,单击“完成”按钮,安装完成。

22 Visual Studio 集成开发环境介绍

221 系统的启动

在操作系统“程序”菜单的“Microsoft Visual Studio 2005”组中选择“Microsoft Visual Studio 2005”,启动 Visual Studio 集成开发环境。

如果是安装后第一次启动,系统会请用户“选择默认环境设置”。如果是一个有经验的使用者,可以在列表中选择自己喜爱的环境设置,但本书后面章节的内容都是基于“常规开发设置”的,读者如果选择其他设置,操作界面可能会与后面章节的内容略有不同。

选择默认环境设置后，系统提示“Microsoft Visual Studio 正在为第一次使用配置环境。这可能要花几分钟。”请耐心等待。配置环境所需的时间与系统内存的大小有关。

Visual Studio 集成开发环境的主界面如图 2-4 所示。



图 2-4　Visual Studio 集成开发环境(一)

界面的顶部是典型的 Windows 菜单及工具栏，左侧的“服务器资源管理器”标签用于在开发环境下连接其他服务器和数据库，“工具箱”标签用于访问工具和控件。右侧的解决方案资源管理器用于浏览文件和与项目相关的类，刚启动时为空。中间是起始页，它包含了新建一个项目和网站或打开一个现存的项目和网站的连接。

222　第一个应用程序

1. 创建网站

VS2005 集成开发环境的操作极为方便，但由于它所提供的功能也非常丰富，因此，不可能在通晓集成开发环境的所有细节后再开始工作，只有在开发过程中逐渐地熟悉。本节先用一个最简单的实例来介绍 VS2005 集成开发环境最基本的使用方法，并帮助用户建立一个使用 VS2005 进行 Web 应用程序开发的初步认识。

按照通常的习惯，先建立一个“Hello World!”程序。但是，在程序中并没有将这个字符串直接写在网页上，而是通过服务器端编程将它写到一个 Label 控件上，并最终在客户端显示。

有两个方法可以创建一个新的网站：

(1) 在集成开发环境的“起始页”上，鼠标单击“最近的项目”标签中的“创建-网站”超链(见图 2-4)。

(2) 在系统主菜单中选择“文件”→“新建”→“网站”。

创建新网站的对话框如图 2-5 所示。

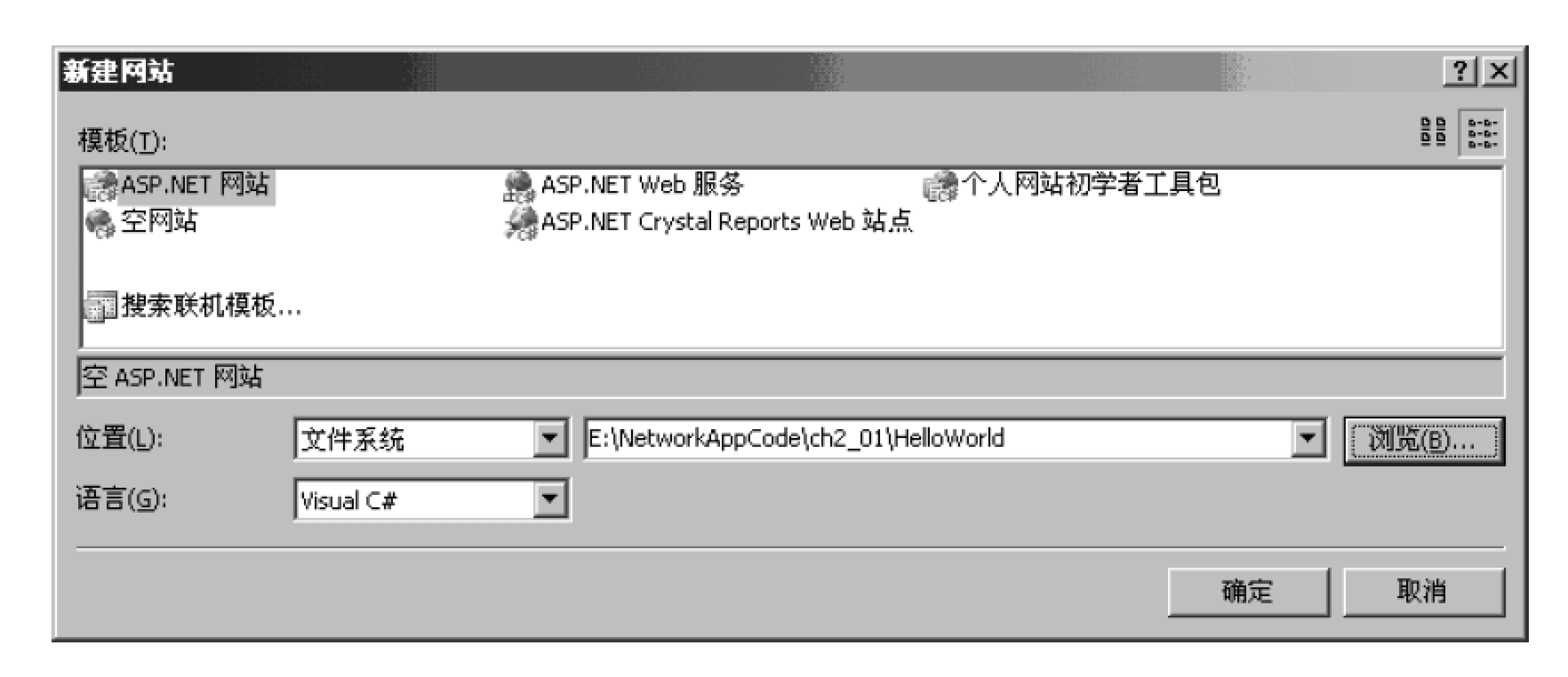


图 2-5 “新建网站”对话框

在“模板”处选择“ASP.NET 网站”。

在“位置”处选择“文件系统”,按“浏览”按钮或直接输入准备存放新网站的子目录名。

提示:在“位置”列表中有 3 个可选项,代表 3 种开发 Web 应用的方式:文件系统、HTTP 和 FTP。

(1) 文件系统

文件系统是默认的选项。使用该选项可以把网站创建到当前物理文件系统上任何可以访问的地方,既可以是本机的一个目录,也可以在网络可访问的其他机器上。

选择文件系统后,VS2005 使用内置的 Web Server 来运行应用程序,而不是使用 IIS,甚至机器上都不必安装 IIS。

通过文件系统创建的网站在发布之前不能够通过浏览器直接浏览,而只能在 VS2005 环境下运行。

使用文件系统的好处是在开发者之间共享开发成果比较容易,只需要将相应的文件目录复制后再用 VS2005 打开即可。本书后面的示例都是基于这种方式的。

(2) HTTP

HTTP 方式指定 IIS 为 Web Server。开发的 Web 应用程序必须是在 IIS 的某个虚拟目录下,这里面涉及一些计算机管理方面的知识,已经超出了本书的范围。好在 VS2005 会自动创建虚拟目录。

(3) FTP

FTP 方式允许开发者在远程计算机上通过 FTP 协议开发 Web 应用程序。

在“语言”处选择“Visual C#”,表示创建的网站以 C# 作为默认的编程语言。按“确定”按钮,VS2005 会在指定的位置创建一个新的网站,并创建新网站的默认主页 Default.aspx。

2. 编程

新网站创建之后，解决方案资源管理器中列出了系统为此网站自动创建的内容（也可以到操作系统相应的目录下面去查看、对比），并在工作区中打开了此网站默认的主页 Default. aspx 供编辑，如图 2-6 所示。

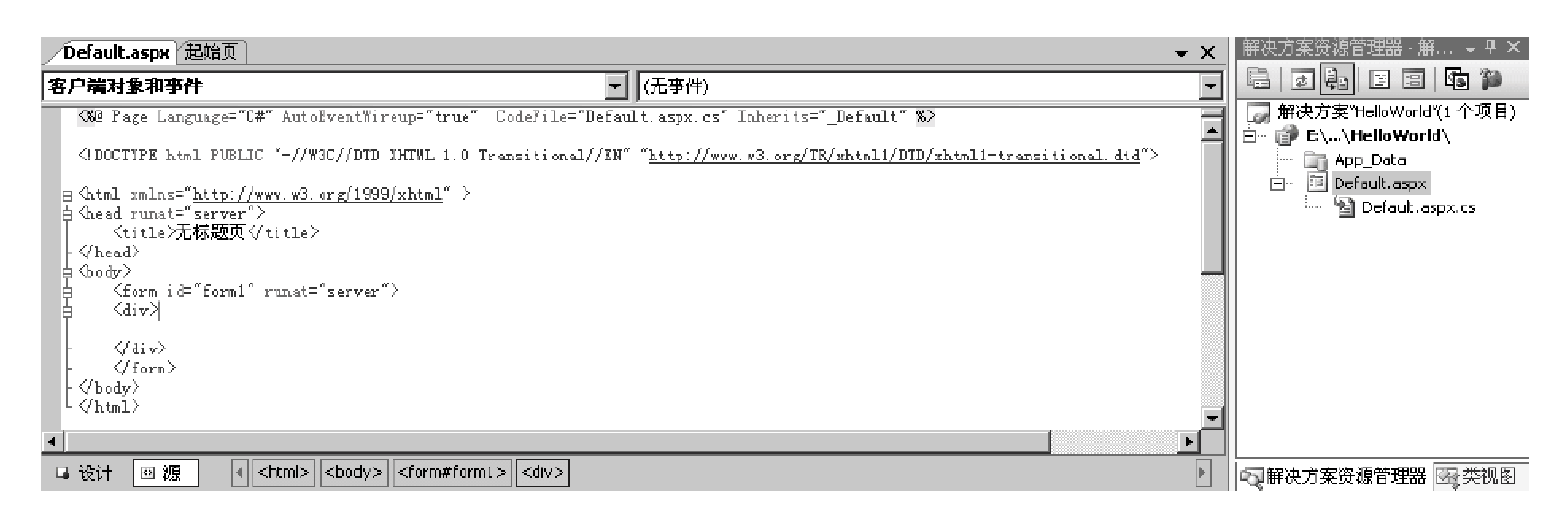


图 2-6　新创建的网站

网页的编辑有两种视图（或状态），设计视图和源视图。在设计状态下，系统显示的是所见即所得的网页效果预览；在源状态下，系统显示的是网页的源代码。

在两种状态下都可以为网页增加控件，方法是将鼠标光标移动到集成环境左侧的“工具箱”标签上，在弹出的工具箱控件列表中选择“Label”控件，用鼠标将其拖动到编辑区适当的位置；如果是在源状态下，则将其拖到<div>和</div>之间。

增加了 Label 控件后，Default. aspx 的源代码段为：

```
<div>
    <asp: Label ID="Label1" runat="server" Text="Label"></asp: Label>
</div>
```

在编辑区中单击鼠标右键，在弹出式菜单中选择“查看代码”，或在集成开发环境右侧的解决方案资源管理器中双击 Default. aspx. cs 文件将其打开，在 Page_Load()函数中输入如下代码：

```
Label1.Text="Hello World!";
```

在集成环境的工具栏中按“全部保存”按钮，保存解决方案。

3. 运行

在解决方案资源管理器中的 Default. aspx 上单击鼠标右键，在弹出菜单中选择“在浏览器中查看”。系统会启动 ASP. NET Development Server，如图 2-7 所示，并自动打开浏览器对刚编辑的网页进行浏览，运行结果如图 2-8 所示。



图 2-7　ASP. NET Development Server 启动后



图 2-8　打开的网页

223　集成开发环境介绍

虽然可以在应用程序的开发过程中逐渐熟悉 VS2005 的集成开发环境(IDE),但在工作开始之前,对 VS2005 集成开发环境先有一个基本的认识还是很有必要的。

图 2-4 给出了 VS2005 集成开发环境的初始状态,图 2-9 给出了正常编程状态下的一个典型布局。从图 2-9 可以看出,VS2005 集成开发环境包括菜单、工具栏、工具箱、可视化窗体设计器、代码编辑窗口、解决方案资源管理器、属性窗口、信息输出窗口等。

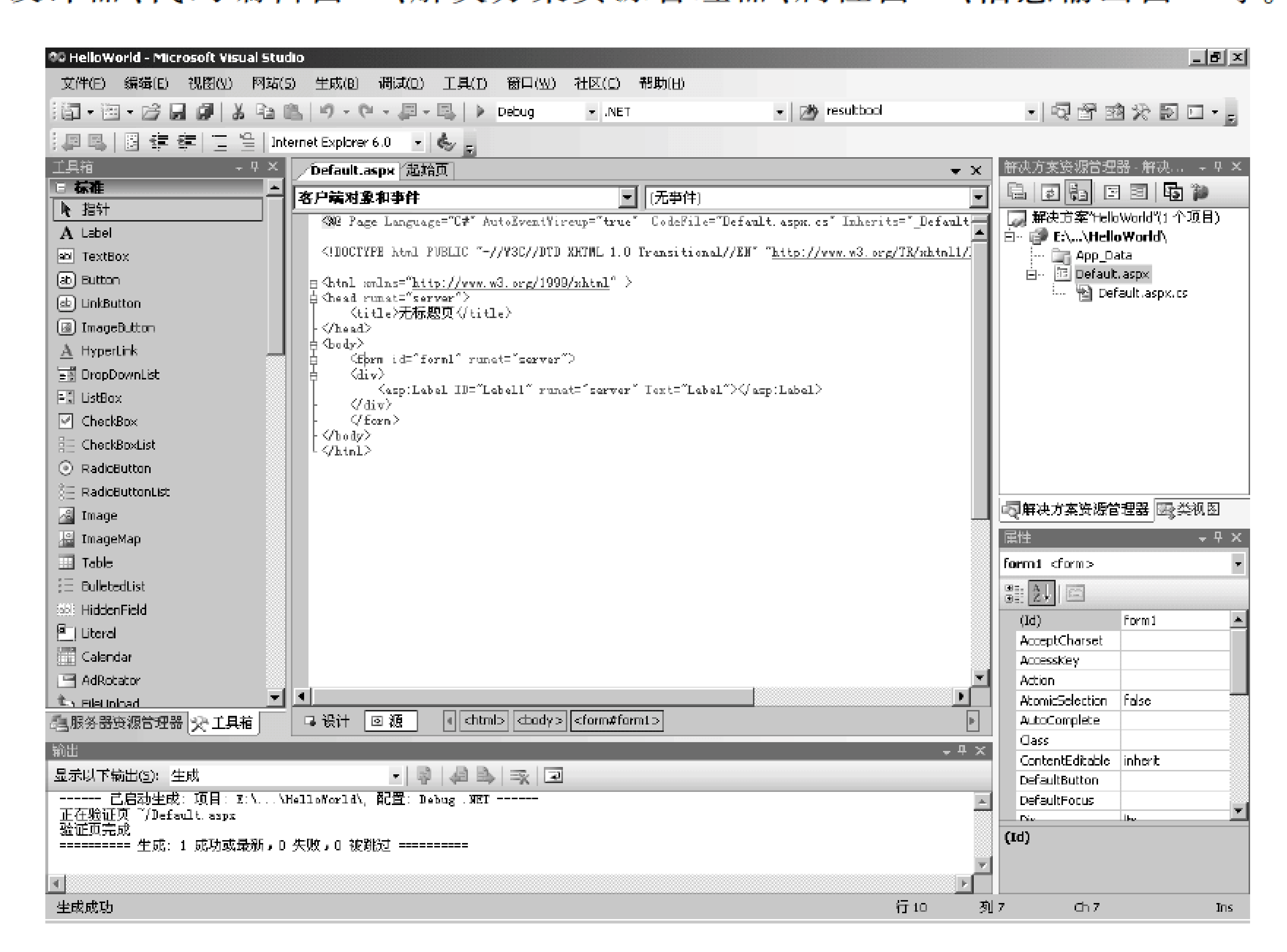


图 2-9　Visual Studio 集成开发环境(二)

熟练掌握 IDE 的各种应用技巧会给开发者的开发效率带来很大提高。

在 VS2005 的 IDE 中,可以直观地将一个控件从工具箱拖放到设计窗口或源代码编辑窗口中。在代码编辑窗口中,智能感知功能将自动提示特定情况下所有可用的成员列表。代码窗口的语法错误在编辑过程中就会是高亮显示,而不必等到编译的时候。

IDE 布局的各部分操作都可以通过鼠标的拖拉、单击、双击等通用操作实现。本小节主要介绍各菜单项的功能。

通过菜单可以执行 IDE 的大部分功能。其中最常用的菜单项都有对应的快捷工具按钮，也可以通过单击快捷按钮来执行这些功能。菜单和工具栏都是上下文相关的，也就是说，在进行不同操作的时候，所看到的菜单和工具栏都可能有所不同。如果正在进行代码编辑，看到的主菜单项可能包括文件、编辑、视图、网站、生成、调试、工具、窗口、社区、帮助等。

下面介绍部分常用菜单项的功能，有些不常用的或具有通常意义的，如关闭、保存等，就不再赘述了。

1. 文件

包含与文件、项目和解决方案有关的命令操作，如表 2-1 所示。

表 2-1 文件的操作命令与功能

菜单项	功 能 说 明
新建	创建新的项目、网站、文件等内容，其中“从现有代码创建项目”可以引导用户将现有文件复制到新建项目中
打开	打开现有的项目、网站、文件等内容，还可以进行不同类型项目之间的转换
添加	将一个新的或已存在的项目或网站添加到已存在的解决方案中
高级保存选项	弹出一个对话框，允许对所使用的编码及行尾进行设置
在浏览器中查看	启动内置的 Web Server(如果需要)，打开浏览器查看当前页面

2. 编辑

包含和代码编辑与查找有关的命令操作，如表 2-2 所示。

表 2-2 编辑的操作命令与功能

菜单项	功 能 说 明
查找和替换	提供了强大的查找与替换功能，每项功能都可以选择应用于当前文件或一定范围内的文件。根据查找内容的不同，这些范围可能包括所有打开的文档、当前项目、当前解决方案、所有组件、自定义组件集等
转到	将光标转到指定的行
将文件作为文本插入	将其他文件的内容作为文本插入到当前光标处
高级	包含了一些只与代码编辑相关的菜单命令，通过这些菜单命令可以： • 将选定行中的空格与制表符互换； • 将选定的文本强制转换为大写或小写； • 删除水平空白； • 查看空白； • 将选定的内容注释或取消注释； • 增加或减少缩进； • 等等

续表

菜单项	功 能 说 明
书签	方便在代码中作标记并在以后快速找到它们
大纲显示	可以展开或收缩任意代码段,从而使文档的结构更加清晰
IntelliSense	智能感知功能是实时的、上下文相关的帮助功能,其中代码完成功能自动完成单词的拼写,从而减少程序员的输入;下拉式列表提供当前对象的所有事件和属性

3. 视图

对 IDE 中所有允许出现的窗口的打开和显示属性进行设置,这些窗口包括服务器资源管理器、解决方案资源管理器、书签窗口、类视图、代码定义窗口、对象浏览器、错误列表、输出、属性窗口、任务列表、工具箱、查找结果等。

4. 网站

包含与解决方案资源管理器中相似的功能,但在解决方案资源管理器中操作会更加方便,如表 2-3 所示。

表 2-3 网站的操作命令与功能

菜单项	功 能 说 明
添加新项	打开“添加新项”对话框,允许用户添加新项到当前项目,如图 2-10 所示
添加现有项	将一个已经存在的项添加到当前项目中
复制网站	把一个网站的所有文件复制到一个新的网站中去
从项目中排除	将当前文件从网站项目中排除出去
启动选项	打开一个对话框,允许对启动操作、服务器和调试器等进行定制

5. 生成

生成或重新生成当前的项目、网站或解决方案。也可以对单独的页进行生成操作。还可以对生成的过程进行配置。

6. 调试

VS2005 允许以调试方式或非调试方式启动应用程序,也允许进行断点的设置、单步执行等典型的调试操作。

7. 工具

VS2005 允许从 IDE 启动一些类似于数据库连接工具这样有用的工具,这些工具大部分是外部工具,包括检查辅助功能、生成本地资源、附加到进程、连接到设备、连接到数

图 2-10 “添加新项”对话框

据库、连接到服务器、代码段管理等。

8. 窗口

包含标准 Windows 应用程序的标准窗口操作，包括自动自动隐藏、新建水平选项卡组、新建垂直选项卡组、关闭所有文档和重置窗口布局等。

9. 社区

没有人能熟悉使用 VS2005 完成 Web 应用程序开发的所有细节，即使再有经验的程序员，也会遇到一些不能独立解决的问题。幸好现在是处在网络时代，可以通过网络与全球成千上万的程序员进行交流。VS2005 提供的社区功能，可以帮助用户快速在互联网上找到想要的资源。用户可以通过社区在相应的论坛上提出问题并得到答复。

使用此菜单下的功能，用户需要有一个可用的 Internet 连接。其实，VS2005 的社区功能只是提供一种帮助，有经验的开发者完全可以在网上自行找到想要的资源。

在 VS2005 的社区菜单下，选择“检查问题状态”项，可直接进入微软中国社区。如果从网上自行进入，其 URL 为 http://bbs.mscommunity.com/forums/。

选择“开发中心”项，可直接进入 MSDN 开发中心。如果从网上自行进入，其 URL 为 http://msdn2.microsoft.com/zh-cn/developercenters/default.aspx。在 MSDN 开发中心，可以迅速找到相关的代码示例，社区站点，技术文章和文档，以及最新活动信息等。

选择“Codezone 社区”项，可直接进入 Microsoft 开发人员网络。如果从网上自行进入，其 URL 为 http://msdn2.microsoft.com/zh-cn/community/aa570297.aspx。在 Codezone 社区，能够更方便地找到与微软开发技术相关的信息和资料。

10. 帮助

对于有经验的程序员来说，在设计开发的过程中如何获得及时有效的帮助也是一个重要的技巧。除了在网上的论坛中进行交流并获得帮助外，使用 VS2005 的联机帮助也是一个重要手段。

VS2005 的“帮助”文档包含在 MSDN Library 中，在使用 VS2005 的过程中，有六种方法可以获得“帮助”。

(1) F1 搜索：按 F1 键可获得上下文相关的帮助搜索功能。

(2) 搜索：使用搜索界面，返回与任何指定的术语或术语集相匹配的所有文档。

(3) 索引：索引可以快速找到本地 MSDN Library 中的文档。

(4) 目录：MSDN Library 目录以分层的树视图结构显示库中的所有主题。

(5) 如何实现：是 MSDN Library 的筛选视图，其中主要包括称为“如何”或“演练”的文档，这些文档说明如何完成特定的开发任务。

(6) 动态帮助：根据代码编辑器中插入点的当前位置，显示到 .NET Framework 和 C#语言的参考文档的链接。

习　　题

1. 打开 Visual Studio 2005 集成开发环境，熟悉并试用菜单、工具栏、工具箱、可视化窗体设计器、代码编辑窗口、解决方案资源管理器、属性窗口、信息输出窗口等内容。

2. 参照 2.2.2 节的介绍，使用 VS2005 集成开发环境创建一个网站，运行并在浏览器中显示“Hello World!”。

3. 在使用 VS2005 创建一个网站项目时，有几种位置选项可以选择？请分别简述。

4. 试总结在开发过程中获得帮助的方法有哪些？

第 3 章 chapter 3

C♯语言基础

在 C♯(读做 C Sharp)之前,C 和 C ++ 一直是最有生命力的程序设计语言。这两种语言为程序员提供了丰富的功能、高度的灵活性与强大的底层控制能力。但与其他 RAD (rapid application development)开发环境相比,如 Visual Basic 或 Delphi,为了实现同样的功能,特别是界面功能,使用 C 和 C ++ 往往需要更长的开发周期和更高的技巧性。

为了解决这一问题,微软推出了 C♯语言。C♯语言是由 C 和 C ++ 派生而来的一种“简单、流行、面向对象、类型安全”的程序设计语言,意在综合 Visual Basic 的高效率和 C ++ 的强大功能。

C♯使得程序员能够在 .NET 平台上快速开发种类丰富、功能强大的应用程序。C♯与 C 和 C ++ 有着很大的相似性,熟悉 C 和 C ++ 的程序员可以很容易地转到 C♯上来。在介绍本章的内容时,我们认为大家已经了解 C 或 C ++ ,并在一定程度上了解面向对象编程的概念。

也正因为 C♯与 C 和 C ++ 的相似性,使得读者可以仅对它做一个简单的了解,就开始 Web 应用程序的实际开发工作。当然,如果要想成为一个真正的使用 ASP. NET 开发 Web 应用程序的专家,还是需要对 C♯作更加全面、深入的学习。

本章首先从一个最简单的 C♯实例开始入手。

3.1 C# 程序实例

3.1.1 创建实例程序

在第 2 章中已经创建了一个称为“Hello World!”的 Web 应用程序。本章再创建一个简单的 C♯控制台应用程序,其功能也是输出“Hello World!”。

在 VS2005 集成开发环境的“起始页”上,鼠标单击“最近的项目”标签中的“创建→“项目”超链,打开“新建项目”对话框,如图 3-1 所示。

在项目类型中选择 Visual C♯,在模板中选择控制台应用程序。输入项目名称为 HelloWorld,解决方案名称也会自动修改为相同的名称。选择适当的项目存储“位置”。按“确定”按钮,VS2005 会自动地在指定位置创建解决方案和项目。有关解决方案和项目的内容将在第 6 章进行详细介绍。



图 3-1 “新建项目”对话框

项目中已经包含了一个称为 Program.cs 的文件(.cs 是 C# 源程序文件的扩展名)。该文件已经在工作区中打开,其代码如下:

```
using System;
using System.Collections.Generic;
using System.Text;

namespace HelloWorld
{
    class Program
    {
        static void Main(string[] args)
        {
        }
    }
}
```

在 static void Main()函数后面的花括号{}中间增加如下 3 行代码:

```
//在主函数中输出文本
Console.WriteLine("Hello World!");
Console.ReadLine();
```

按 F5 键或鼠标单击快捷工具栏中的“启动调试”按钮,执行程序。

本程序在一个弹出的窗口中显示“Hello World!”,并等待用户输入;用户按回车键,程序退出,窗口关闭。

3.1.2 代码分析

本节对 3.1.1 小节的程序代码做一个简单分析，使读者能够先对 C# 语言程序有一个概要的了解。

和 C 语言、C++ 语言一样，C# 语言对大小写也是非常敏感的。

为了使用户编程更加方便，.NET Framework 中提供了众多有用的预定义类供用户使用，为了方便这些类的管理和引用，引入了命名空间的概念。例如 Console 就是一个预定义的用于输入、输出的类，该类在 System 命名空间中定义。引用该类的原始方法是 System.Console；但如果在程序的开头处使用 using 关键字引用了 System 命名空间，在程序中就可以直接引用 Console 类了，这样可以给代码输入带来方便。命名空间是分级管理的，引用时各级之间用圆点(.)隔开。

在程序代码的前 3 行，分别引用了 3 个最常用的命名空间。对于本程序而言，其实只引用 System 命名空间就够了。

在后续的代码中，使用 namespace 关键字声明了一个与项目同名(HelloWorld)的命名空间，便于组织本项目代码。在 HelloWorld 命名空间中又声明了一个称为 Program 的类。

C# 是一个“纯”的面向对象的语言。C# 中不再有全局变量和全局函数，任何变量和函数都必须属于一个类，包括程序的主函数 Main()，在本例中，Main()就是 Program 的一个静态方法。在 C# 程序中，程序的执行总是从 Main()方法开始的。一个程序中不允许出现两个或两个以上的 Main()方法。

与 C、C++ 一样，C# 语言的程序代码块被包含在花括号{}中；C# 也可以使用双斜杠(//)来进行单行注释或使用分割符/* 和 */来进行多行注释。

在 C# 语言中，程序的输入输出功能是通过 Console 类来完成的。Console 是在命名空间 System 中预定义的一个类。在上述代码中，使用了 Console 类的两个最基本的方法：WriteLine 和 ReadLine。有关 C# 语言的控制台输入输出将在 3.3.3 节进一步介绍。

3.2 数据类型

在 .NET 中，任何类型都是“类”。

C# 语言支持的数据类型包括两大类：值类型和引用类型。值类型通常被分配在堆栈上，它的变量直接包含变量的实例，使用的效率比较高。引用类型总是分配在托管堆上，引用类型的变量通常仅包含一个指向实例的指针，系统通过该指针来引用其实例。

值类型包括整数类型、布尔类型、实数类型、字符类型、结构类型和枚举类型等；引用类型包括类和数组等。

3.2.1 值类型

1. 整数类型

C#语言中有多种整数类型,如表 3-1 所示。

表 3-1 C#语言中的整数类型

类 型	名 称	范 围	大 小
sbyte	短字节型	−128～127	有符号 8 位整数
byte	字节型	0～255	无符号 8 位整数
short	短整型	−32 768～32 767	有符号 16 位整数
ushort	无符号短整型	0～65 535	无符号 16 位整数
int	整型	−2 147 483 648～2 147 483 647	有符号 32 位整数
uint	无符号整型	0～4 294 967 295	无符号 32 位整数
long	长整型	−9 223 372 036 854 775 808～9 223 372 036 854 775 807	有符号 64 位整数
ulong	无符号长整型	0～18 446 744 073 709 551 615	无符号 64 位整数

2. 布尔类型

布尔(bool)类型是用来表示"真"和"假"这两个概念的,分别采用 True 和 False 两个值来表示。可将布尔值赋给 bool 变量,也可以将 bool 表达式赋给 bool 变量。在 C 和 C++ 中用 0 来表示假,其他任何非 0 的值都表示真,这种情况在 C#中已经被彻底改变。下面是一个使用布尔型变量的例子:

```
bool b=True;
char c='A';
Console.WriteLine(b);
bool d=(c>'a');
Console.WriteLine(d);
```

运行结果为:

```
True
False
```

3. 实数类型

C#语言中的实数类型如表 3-2 所示。

表 3-2　C#语言中的实数类型

类　型	名　称	大致范围	特　征
float	单精度浮点数	±1.5e−45～±3.4e38	32位数据，精度7位
double	双精度浮点数	±5.0e−324～±1.7e308	64位数据，精度15～16位
decimal	十进制类型	±1.0e−28～±7.9e28	128位数据，精度28～29位

4. 字符类型

C#语言支持的字符类型采用Unicode字符集，一个Unicode的标准字符长度为16位，范围为U+0000～U+ffff，用它可以来表示世界上大多数语言。下面的例子说明了一个字符型变量：

```
char c='A';
```

C#语言也支持转义字符的使用，转义字符的规定与C语言基本相同。

5. 结构类型

与C语言和C++语言相同，C#语言也可以用struct来说明结构类型。关于结构类型的详细介绍见3.5节。

6. 枚举类型

与C语言和C++语言一样，C#语言也可以用enum来说明枚举类型，如：

```
enum WeekDay
{
    Sunday,Monday,Tuesday,Wednesday,Thursday,Friday,Saturday
};
WeekDay day=WeekDay.Tuesday;
Console.WriteLine("Tuesday={0}",day);
int x=(int)WeekDay.Sunday;
Console.WriteLine("Sunday={0}",x);
```

运行结果为：

```
Tuesday=Tuesday
Sunday=0
```

可以看出，在C#语言中，枚举类型值可以直接输出，输出结果为其枚举标识符本身，这一点与C语言有所不同。枚举值可以与整数值相互转换，转换的规定与C语言相同。

3.2.2 引用类型

C#语言中的另一大数据类型是引用类型。“引用”这个词在这里的含义是该类型的

变量不直接存储所包含的值，而是指向它所要存储的值，也就是说，引用类型存储实际数据的地址。C#语言中的引用类型有类、代表、数组和接口等，在此先简单介绍类和数组。

1. 类

类是面向对象编程的基本单位，是一种包含数据成员、函数成员和嵌套类型的数据结构。类的数据成员有常量、域和事件，函数成员包括方法、属性、索引指示器、运算符、构造函数和析构函数。

类和结构同样都包含了自己的成员，但它们之间最主要的区别在于：类是引用类型而结构是值类型。

类支持继承机制，通过继承，派生类可以扩展基类的数据成员和函数方法，从而达到代码重用和设计重用的目的。

如果对某个类定义了一个变量，则称该变量为类的一个实例。

关于类的详细介绍见3.5节。

2. 数组

在C#语言中，数组的声明格式为：

```
type[] arrayName;
```

例如，声明一个整数数组：

```
int[] arr;
```

在定义数组的时候可以预先在[]中指定数组元素的个数。数组的元素个数也可以通过数组的Length属性获得。

对数组元素的引用与C语言相同，采用“数组名[下标]”的形式。C#语言中的数组元素的下标也是从0开始。下面是一个使用数组的例子：

```
int[] arr=new int[5];
for (int i=0; i<arr.Length; i++)
    arr[i]=i* i;
for (int i=0; i<arr.Length; i++)
    Console.WriteLine("arr[{0}]={1}",i,arr[i]);
```

可以看出，由于数组是引用类型，因此不能只说明一个数组变量就对它进行存取，必须先使用new运算符创建一个数组的实例。

这个程序创建了一个基类型为int的一维数组，初始化后逐项输出。其中arr.Length表示数组元素的个数，运行结果为：

```
arr[0]=0
arr[1]=1
arr[2]=4
```

```
arr[3]=9
arr[4]=16
```

在 C 语言中，数组仅仅是连续存储的一组数据，而 C# 语言中数组则是基于类的，这就提供了更强的功能和更高的安全性。如 C# 语言的数组提供了 Length 属性方便编程；在 C# 语言中，数组的越界存取是被禁止的。

3.3 C# 基本操作

3.3.1 变量和常量

1. 变量

C# 同样用变量(variable)来保存数据，前一节已经给出了很多变量的例子，在此再做一个简单的总结。C# 中变量的命名原则与 C 和 C++ 相似但略有不同。

(1) 变量名必须以字母、下划线或@开头。

(2) 其后的字符可以是字母、数字和下划线，而不能包含空格、标点符号、运算符等其他符号。

(3) 变量名不能与 C# 中的关键字名称相同。

(4) 变量名不能与 C# 中的库函数名称相同。

在 C# 中，变量分为七种类型，它们分别是静态变量(static variables)、非静态变量(instance variables)、数组元素(array elements)、值参数(value parameters)、引用参数(reference parameters)、输出参数(output parameters)和局部变量(local variables)。

2. 常量

常量就是其值固定不变的量。从数据类型角度来看，常量的类型可以是任何一种值类型或引用类型。常量的声明使用 const 关键字，下面是一个常量声明的例子：

```
public const int m=10,n=20;
```

3.3.2 装箱和拆箱

object 类是所有其他类型的基类，C# 语言中的所有类型都直接或间接地从 object 类中继承。因此，对一个 object 的变量可以赋予任何类型的值。

装箱(boxing)和拆箱(unboxing)是在 C# 语言的类型系统中提出的一个核心概念。装箱和拆箱机制使得在 C# 类型系统中任何值类型、引用类型和 object 对象类型之间都可以进行转换。

装箱和拆箱使值类型能够被视为对象。对值类型装箱将把该值类型打包到 object 引用类型的一个实例中。拆箱将从对象中提取值类型。下面是一个装箱、拆箱及使用 object 对象的例子：

```
int y=25;
object obj1;
obj1=y;
int z=(int)obj1;
Console.WriteLine(z);
object obj2='A';
        Console.WriteLine(obj2);
```

在此示例中，整型变量 y 被“装箱”并赋值给对象 obj1，拆箱后再将其值提取到整型变量 z 中。程序的运行结果为：

```
25
A
```

可以看出拆箱过程正好是装箱过程的逆过程。值得注意的是，装箱转换和拆箱转换必须遵循类型兼容原则。

3.3.3 控制台输入输出

C#语言程序所有的控制台输入输出功能都是通过 Console 来完成的。Console 是在 System 命名空间中已经定义好的一个类，可以使用它的公共方法来完成控制台的输入输出功能。Console 类常用的公共方法见表 3-3。

表 3-3 Console 类常用的公共方法

方法名称	说明
Beep	通过控制台扬声器播放提示音
Read	从标准输入流读取下一个字符
ReadLine	从标准输入流读取下一行字符
Write	将指定值以文本表示形式写入标准输出流
WriteLine	将指定的数据(后跟当前行结束符)写入标准输出流

下面是一个完成控制台输入输出的例子：

```
Console.WriteLine("Please enter a word: ");
string word=Console.ReadLine();
Console.WriteLine("The word is : {0}!",word);
```

上述代码的执行结果为：

```
Please enter a word:
Happy<Enter>
The word is : Happy!
```

在上述代码中，先调用 Console 的 WriteLine 方法显示一行提示文字，再调用 ReadLine 方法等待用户输入，将输入的一行文本存放到字符串变量 word 中，再调用

WriteLine 输出结果。

Console 中另外两个常用的输入输出方法是：Read 和 Write。它们和 ReadLine 与 WriteLine 的不同之处在于，ReadLine 和 WriteLine 执行时相当于在显示时多加了一个回车键，而使用 Read 和 Write 时则光标不会自动转移到下一行。

3.3.4 字符串处理

大多数实际应用中都会经常用到字符串的处理。C# 语言所提供的字符串类具有丰富的字符处理功能。

1. 使用 string

C# 语言字符串是使用 string 关键字声明的一个字符数组。声明字符串常量使用双引号，如：

```
string s="Hello,World!";
```

字符串作为一个类，提供了很多有用的公共方法进行通用的字符串操作，表 3-4 列出了其中最常用的部分。

表 3-4　string 类常用的公共方法

方法名称	说　明
CompareTo	将此实例与指定的对象或 string 进行比较，并返回比较结果
Concat	将一个或多个字符串连接在一起
Contains	返回一个布尔值，表示此字符串是否包括另一个 string 对象。如： b=str1.Contains(str2);
Copy	字符串复制。如： str2=string.Copy(str1);
CopyTo	将指定数目的字符从此字符串中的指定位置复制到 Unicode 字符数组中的指定位置
EndsWith	判断此字符的末尾是否与指定的字符串匹配
Equals	判断两个字符串是否相等
Format	将指定字符串中的每个格式项替换为相应对象的值的文本
IndexOf	在此字符串中查找子字符串第一次出现的位置。如： int i=str1.IndexOf(str2);
IndexOfAny	在此字符串中查找指定 Unicode 字符数组中的任意字符第一次出现的位置
Insert	将一个字符串插入到另一个字符串的指定索引位置。如： str1=str1.Insert(5,str2);
IsNullOrEmpty	判断指定的 string 对象是空引用还是 Empty 字符串
Join	在指定 string 数组的每个元素之间串联指定的分隔符 string，从而产生单个串联的字符串

续表

方法名称	说　明
LastIndexOf	在此字符串中查找子字符串最后一次出现的位置
LastIndexOfAny	在此字符串中查找指定 Unicode 字符数组中的任意字符最后一次出现的位置
PadLeft	在当前字符串的左侧填充空格或指定字符，使当前字符串达到指定的总长度，以达到右对齐的效果
PadRight	在当前字符串的右侧填充空格或指定字符，使当前字符串达到指定的总长度，以达到左对齐的效果
Remove	从此实例中删除指定个数的字符
Replace	将当前字符串中的所有指定子串替换为其他指定子串
Split	将一个字符串去掉指定分隔符后，分裂成 string 数组
StartsWith	判断当前字符串的开头是否与指定的字符串匹配
Substring	从此字符串中取子字符串。如： s2=s1.Substring(2,8);
ToCharArray	将此字符串中的字符复制到 Unicode 字符数组
ToLower	返回此字符串转换为小写形式的副本
ToString	将此实例的值转换为 string
ToUpper	返回此字符串转换为大写形式的副本
Trim	从此字符串的开始位置和末尾移除一组指定字符的所有匹配项
TrimEnd	从此字符串的结尾移除数组中指定的一组字符的所有匹配项
TrimStart	从此字符串的开始位置移除数组中指定的一组字符的所有匹配项

上述公共方法都是通过点操作符来调用，其语法格式为：

字符串.方法名(参数表)

下面对表 3-4 中的一些常用公共方法做一个介绍。

字符串对象提供的 CompareTo()方法用于比较两个字符串，可以得到某个字符串是否小于、等于或大于另一个字符串。使用方法为：

```
int i=stringA.CompareTo(stringB)
```

其返回值是一个整数，反应两个字符串间的大小关系：

i 小于零：stringA 小于 stringB；

i 等于零：stringA 等于 stringB；

i 大于零：stringA 大于 stringB。

如：

```
string s3="red";
string s4="green";
```

```
int r= s3.CompareTo(s4);
if (r> 0)
     Console.WriteLine("red> green");
```

输出为：

```
red> green
```

如果仅仅是要知道两个字符串是否相等，最简单方法是使用==和!=运算符。如：

```
string s3= "red";
string s4= "green";
if (s3= = "red")
     Console.WriteLine("red= = red");
if (s3 != s4)
     Console.WriteLine("red!= green");
```

输出为：

```
red= = red
red!= green
```

对于熟悉 Java 语言的程序员来说，在需要比较两个字符串是否相等时可能更喜欢使用 Equals 方法。C#语言也提供了 Equals 方法，如：

```
string s4= "green";
if (s4.Equals("green"))
     Console.WriteLine("green= = green");
```

输出为：

```
green= = green
```

如果不是进行字符串的完整比较，而只是判断当前字符串的开头是否与指定的字符串匹配，可使用 StartsWith 方法。如：

```
s3= "ABCDEF";
//精确比较
Console.Write("ABCDEF ");
if (!s3.StartsWith("abc")) Console.Write("doesn't start");
else Console.Write("starts");
Console.Write(" with abc.\n");
//忽略大小写
Console.Write("IgnoreCase: ABCDEF ");
if (!s3.StartsWith("abc",StringComparison.CurrentCultureIgnoreCase))
     Console.Write("doesn't start");
else Console.Write("starts");
Console.Write(" with abc.\n");
```

输出为：

```
ABCDEF doesn't start with abc.
IgnoreCase: ABCDEF starts with abc.
```

两个字符串的连接还可以直接采用+运算符，如：

```
string s1="Hello,";
string s2="World!";
s3=s1+s2;
Console.WriteLine(s3);
s1+=s2;
Console.WriteLine(s1);
```

使用 Concat 方法更灵活，可以连接多个字符串或对象。如：

```
Console.WriteLine(string.Concat(s1,s2,s3));
```

上面两段程序连续执行的输出为：

```
Hello,World!
Hello,World!
Hello,World!World!Hello,World!
```

空字符串是字符串的特殊情况，往往需要进行特殊的处理。因此，判断字符串是否为空就显得特别重要。

字符串等于 null 和字符串等于""是有区别的。字符串等于 null 表示这个字符串根本就不存在；而字符串等于""则表示存在一个字符串，只不过它的字符个数为 0。而在实际应用中，又往往需要对这两种情况进行同样的处理，这时就可以用 IsNullOrEmpty 方法进行判断。如：

```
if (null=="")
    Console.WriteLine("NULL==\"\"");
else
    Console.WriteLine("NULL!=\"\"");
s1=null;
if (String.IsNullOrEmpty(s1))
    Console.WriteLine("S1 is null.");
s1="";
if (String.IsNullOrEmpty(s1))
    Console.WriteLine("S1 is empty.");
```

输出为：

```
NULL!=""
S1 is null.
S1 is empty.
```

有时往往需要向字符串的后面或前面填充字符，使其达到指定长度，如想在输出时

得到左对齐或右对齐的效果，可使用字符串的 PadLeft 或 PadRight 方法。如：

```
s1="CSharp";
Console.WriteLine(s1.PadLeft(15));
Console.WriteLine(s1.PadLeft(15,'.'));
```

输出为：

```
         CSharp
.........CSharp
```

如果要去掉字符串开头或结尾的指定字符，则需要使用 Trim、TrimEnd 和 TrimStart 等方法。

在 C#语言中，很多类型的对象都提供了 ToString 方法，用于将其值转换为字符串。如：

```
int n=188;
string msg="The number is "+n.ToString();
Console.WriteLine(msg);
```

输出为：

```
The number is 188
```

注意：字符串对象也提供 ToString 方法，但执行字符串对象的 ToString 方法不进行实际转换。

字符串类中还有一个公共属性 Length，可以用来获取字符串的长度。字符串也可以像数组一样用[]加下标来获取指定位置的字符。在字符串中也可以使用转义符，如“\n”(换行)和“\t”(制表符)等。下面的例子表现了这些特性：

```
Console.Write("Type a string : \n");
string aString=Console.ReadLine();
for (int i=0; i<aString.Length; i++)
    Console.WriteLine("{0}",aString[i]);
```

运行结果为：

```
Type a string :
Xu<Tab>        Bo<Enter>
X
u

B
o
```

关于字符串公共方法更多更详细的说明可以查阅 VS2005 的联机帮助。

2. 使用 StringBuilder

string 对象是“不可变的”，即它们一旦创建就无法更改。对字符串进行操作的方法

实际上返回的是新的字符串对象。因此，出于性能方面的原因，大量的连接或其他涉及字符串的操作应当用 StringBuilder 类执行，如：

```
System.Text.StringBuilder s5=new System.Text.StringBuilder();
s5.Append("I");
s5.Append("love");
s5.Append("you!");
string s6=s5.ToString();
Console.WriteLine(s6);
```

上述代码的执行结果为：

```
I love you!
```

StringBuilder 类创建了一个字符串缓冲区，用于在程序执行大量字符串操作时提供更好的性能。StringBuilder 还允许修改字符串中的个别字符，这是 string 类型所不支持的。例如，下面代码在不创建新字符串的情况下更改了一个字符串的内容：

```
System.Text.StringBuilder s7=
    new System.Text.StringBuilder("congratulations!");
s7[0]='C';
System.Console.WriteLine(s7.ToString());
```

上述代码的执行结果为：

```
Congratulations!
```

3.4 流程控制

C＃语言的流程控制语句与 C 语言基本相同。

在学习第一种编程语言时，流程控制语句总是其中最重要的内容。但是，当使用过几种编程语言之后，有经验的程序员大多有过这种感触：当初被当做难点的流程控制语句其实是最简单的了，因为每一种语言的这部分都非常相似。本书认为读者已经具有一定的 C 语言或 C ++ 语言的基础，因此，这部分内容只做一个简单介绍。

3.4.1 条件语句

1. if 语句

if 语句是条件选择语句，它通过判断给定的条件是否为真，来决定所要执行的操作。if 语句的一般形式如下：

```
if(布尔表达式)
    语句 1
else
```

```
    语句 2
```

if 语句的执行过程是：首先计算 if 后面括号内布尔表达式的值，如果它的值为真，就执行语句 1；如果表达式的值为假，就执行语句 2。在 if 语句中，可以省略 else 和语句 2 部分，其形式为：

```
if(布尔表达式)
    语句
```

2. switch 语句

switch 语句用于多分支选择，它的一般形式如下：

```
switch(控制表达式)
{
    case 常量表达式 1: 语句组 1
    case 常量表达式 2: 语句组 2
    …
    case 常量表达式 n: 语句组 n
    default:          语句组 n+1
}
```

switch 语句中控制表达式的数据类型可以是 sbyte、byte、short、ushort、int、uint、long、ulong、char、string 或枚举类型。每个 case 标签中的常量表达式必须属于或能隐式转换成控制表达式类型。

switch 语句的执行过程是：首先计算 switch 后面圆括号内控制表达式的值，然后依次与各个 case 标签后面的常量表达式的值相比较，若一致就执行该 case 标签后面的语句组，直到遇到 break 语句。如果有两个或两个以上 case 标签中的常量表达式值相同，编译时将会报错。如果表达式的值与所有常量表达式的值都不相等，则转向 default 标签后面的语句组去执行，如果没有 default 部分，则不执行任何语句，而直接转到 switch 语句后面的语句去执行。switch 语句中最多只能有一个 default 标签。

C# 语言的 switch 语句与 C 语言和 C++ 语言有如下不同。

① C 语言和 C++ 语言允许 switch 语句中 case 标签后不出现 break 语句，但 C# 语言不允许这样，它要求每个标签项后使用 break 语句或跳转语句 goto，即不允许从一个 case 自动遍历到其他 case，否则编译时将报错。如果想要像 C 语言和 C++ 语言那样，执行完一个语句组后继续执行其他的语句，只需要显式地加入 goto case label（跳至标签语句执行）或 goto default（跳至 default 标签执行）语句即可。

② C# 语言可以把字符串当成常量表达式来使用，所以 switch 语句的控制类型可以是 string 类型。

3.4.2 循环语句

1. while 循环语句

while 循环语句的一般形式如下：

```
while(布尔表达式)
    语句(即循环体)
```

该语句的执行过程是：先计算 while 后面圆括号内布尔表达式的值，如果布尔表达式的值为真，则执行后面的循环体语句，然后再次计算布尔表达式的值；重复上述过程，直到布尔表达式的值为假时退出循环。

2. do-while 循环语句

do-while 循环语句的一般形式如下：

```
do
    语句(即循环体)
while(布尔表达式);
```

该语句的执行过程是：先执行循环体语句，再计算 while 后面圆括号内布尔表达式的值，如果其值为真，则再次执行循环体；如此重复，直到布尔表达式的值为假时结束循环。

3. for 循环语句

for 循环语句的一般形式如下：

```
for(初始化;条件;循环)
    语句(即循环体)
```

for 后面括号中的三部分都是可选的，其中初始化和循环还可以由多个语句(用逗号隔开)组成。“初始化”是循环变量赋初值部分，通常为赋值语句。“条件”是循环控制条件，为布尔表达式。“循环”是循环变量的修改部分，用来表达循环变量的增量，通常是赋值语句，常用自加、自减运算。语句部分为循环体，可以是一条语句，也可以是复合语句和空语句。

for 语句的执行过程是：先执行初始化部分。再计算条件表达式的值，若该值为假，则退出循环；若为真，则执行循环体。然后执行“循环”部分，对循环变量进行修改后再计算条件表达式，若为真，再一次执行循环体。如此重复，直到条件表达式的值为假时退出循环。

4. foreach 语句

foreach 语句是在 C# 语言中新引入的，C 和 C++ 语言中没有这个语句。它表示收集一个集合中的所有元素，并针对每个元素执行一次循环体语句。

foreach 语句的一般格式如下：

```
foreach(类型 标识符 in表达式)
    循环体语句
```

其中类型(type)和标识符(identifier)用来声明循环变量，表达式(expression) 对应集合。

每次循环，先从集合中取一个元素赋给循环变量，再执行一次循环体语句；当依次处理完集合中的所有元素后，循环结束。在这里，循环变量是一个只读型局部变量。如果试图改变它的值，或将它作为一个 ref 或 out 类型的参数传递，都将引发编译时错误。

foreach 语句中的表达式（expression）结果必须是集合类型。如果该集合的元素类型与循环变量类型不一致，则必须有一个显式定义的从集合中的元素类型到循环变量类型的转换。

下面是一个在数组上使用 foreach 语句的例子：

```
int[] Fibonacci=new int[] {0,1,2,3,5,8,13};
foreach (int i in Fibonacci)
{
    System.Console.WriteLine(i);
}
```

运行结果为：

```
0
1
2
3
5
8
13
```

在本例中先初始化了一个 Fibonacci 数列放在了数组中。然后使用 foreach 语句显示其中的每个元素。

从本例可以看出，数组类型是支持 foreach 语句的，属于集合类型。对于一维数组，执行顺序是从下标为 0 的元素开始一直到数组的最后一个元素；对于多维数组，元素下标的递增是从最右边那一维开始的，以此类推。

C＃语言中包括很多有用的集合类型，如数组、ArrayList、哈希表、字典、堆栈、队列等，本书不再对这些集合类型进行详细介绍。下面再给出一个在哈希表上使用 foreach 语句的例子：

```
Hashtable myHT=new Hashtable();
myHT.Add(0,"zero");
myHT.Add(1,"one");
myHT.Add(8,"eight");
foreach (DictionaryEntry de in myHT)
    Console.WriteLine("Key={0}\tValue={1}",de.Key,de.Value);
```

运行结果为：

```
Key=8    Value=eight
Key=1    Value=one
Key=0    Value=zero
```

5. 循环的退出

在上面介绍的 4 种循环语句中，都可以使用 break 语句结束循环，继续执行循环语句后面的语句；也可以用 continue 语句来停止本次循环体语句的执行，继续进行下一轮循环。

3.4.3 异常处理语句

在编写程序时，需要考虑到各类不可预期的事件，比如被 0 除、内存不够、磁盘出错、网络资源不可用、数据库无法访问等，C＃语言具有完备的异常处理功能，提供了处理程序运行时出现的任何异常情况的方法。

C＃语言的异常处理机制与 C ++ 语言非常相似，异常可以在两种不同的方式下被引发：

- 在程序中使用 throw 语句，主动、即时地抛出异常；
- 语句和表达式执行过程中激发了某个异常的条件，使得操作无法正常结束，从而引发异常。

使用 throw 语句抛出一个异常，其格式为：

```
throw [表达式];
```

异常处理使用 try、catch 和 finally 关键字来尝试可能引发异常的操作，处理失败，以及在事后清理资源，其一般形式为：

```
try
{
    执行部分
}
catch (异常类型 异常标识符)
{
    异常处理
}
finally
{
    必要处理
}
```

从上面的形式可以看出，异常处理分为 3 个部分，其中第 2、3 部分是可以省略的。

在 C＃语言中，程序中的运行时错误使用一种称为“异常”的机制在程序中传播。异常由遇到错误的代码引发，又能够更正错误的代码捕获。异常可由 .NET Framework 公共语言运行库（CLR）或由程序中的代码引发。一旦引发了一个异常，未捕获的异常由系统提供的通用异常处理程序处理，该处理程序会显示一个对话框。

程序执行“执行部分”，如果未发生异常，则不需要进行异常处理；如果发生异常，这个异常就会在调用堆栈中往上传播，直到找到指针对它的 catch 语句。catch 子句可以有

多个,分别捕获并处理不同种类的异常类型。未捕获的异常由系统提供的通用异常处理程序处理,程序将停止执行,并显示一条错误信息。

无论是否引发了异常,finally 块中的“必要处理”代码总会被执行,因此,可以将用于释放资源的代码放在此处。

下面是一个异常处理的示例。

```
short Fact=1;
short n,i;
n=Convert.ToInt16(Console.ReadLine());
try
{
    for (i=1; i<=n; i++)
        checked {Fact* =i;}
    Console.WriteLine("{0}!={1}",n,Fact);
}
catch (Exception e)
{
    Console.WriteLine("程序执行发生异常:"+e.Message);
}
finally
{
    Console.WriteLine("finally块中的代码总会被执行。");
}
```

说明:这是一个求阶乘的例子,但其中的整数都使用了 short 类型,这样就增大了引起溢出的可能性。如果输入的值为 3,无溢出,则程序的运行结果为:

```
3↙
3!=6
finally块中的代码总会被执行。
```

如果输入的值为 10,则产生溢出,程序的运行结果为:

```
10↙
程序执行发生异常:算术运算导致溢出。
finally块中的代码总会被执行。
```

如果 catch 子句中指定了一个异常类型,则必须是 System. Exception 类型或它的派生类型。如果同时指定了类型和标识符,就是声明了一个异常变量。异常变量相当于一个作用范围为整个 catch 块的局部变量。在 catch 块的执行过程中,异常变量描述了当前正在处理的异常。如果想引用异常对象(其中包括很多重要的错误信息),就必须定义异常变量。

checked 和 unchecked 操作符用于整型算术运算时控制当前环境中的溢出检查。当算术运算产生一个目标类型无法表示的大数时,在使用了 checked 操作符的表达式中,会

抛出溢出异常；而在使用了 unchecked 操作符的表达式中，返回值被截掉不符合目标类型的高位，而不抛出异常。

如果变量表达式没有包括任何 checked 或 unchecked 操作符，溢出时是否会抛出异常，依据于外部因素，如编译器状态、执行环境参数等，而对于一个常量表达式而言，总是默认为进行溢出检查。

使用了 unchecked 操作符后，溢出的发生不会导致编译错误，但这往往会出现一些不可预期的结果，所以使用 unchecked 操作符要特别小心。

当 try 语句执行完以后，finally 块中的语句必将被执行，不论是否会发生由以下原因导致的程序控制转移：

- 普通操作；
- 执行 break、continue、goto 或 return 语句；
- 将异常传播到语句之外。

3.5 类和结构

C#语言支持面向对象的所有关键概念：封装、继承和多态性。整个 C#语言的类模型是建立在.NET 虚拟对象系统之上的，对象模型是基础架构(.NET Framework)的一部分，而不再是编程语言的一部分。

3.5.1 定义类和结构

类和结构都可以包含构造函数、常数、字段、方法、属性、索引器、运算符、事件和嵌套类型等，但结构是值类型，而类是引用类型。在使用和申明结构、类时，需要注意下面几点。

(1) 不能声明结构的默认(无参数)构造函数，系统总是提供默认构造函数将结构成员初始化为它们的默认值，不需要再显式地声明。

(2) 在结构中，不能初始化实例字段。

(3) 结构不能像类那样继承。

(4) C#语言仅允许单个继承。也就是说，类只能从一个基类继承实现。但是，一个类可以实现一个以上的接口，即类最多继承一个类，但可以继承多个接口。

(5) 结构类型永远不会是抽象的(抽象的概念将在 3.5.6 节中介绍)。

(6) 使用 new 运算符创建结构对象时，将创建该对象的实例，并且调用适当的构造函数。

下面是一个定义类和结构的例子：

```
public class NameClass
{
    //定义成员变量
    private string m_Name;
```

```
        //定义方法 GetName()
        public string GetName()
        {
            return m_Name;
        }

        //定义方法 SetName()
        public void SetName(string Name)
        {
            m_Name=Name;
        }
    }

    public struct NameStruct
    {
        //定义成员变量
        private string m_Name;

        //定义方法 GetName()
        public string GetName()
        {
            return m_Name;
        }

        //定义方法 SetName()
        public void SetName(string Name)
        {
            m_Name=Name;
        }
    }

    class Program
    {
        static void Main(string[] args)
        {
            //对象初始化
            NameClass pcName=new NameClass();
            pcName.SetName("Gao Yi");
            Console.WriteLine("My name is "+pcName.GetName().ToString());

            //结构初始化
            NameStruct psName=new NameStruct();
            psName.SetName("Li Lei");
            Console.WriteLine("Your name is "+psName.GetName().ToString());
```

```
    }
}
```

程序中分别创建了一个 NameClass 和 NameStruct 实例,并分别调用了 SetName()方法设值和 GetName()方法取值。程序运行结果为:

```
My name is Gao Yi
Your name is Li Lei
```

在此例中,类和结构还没有太大的差别。

3.5.2 定义属性

在类和结构中,都可以定义属性。属性的定义通常包含两个部分:

(1) 专用数据成员的定义。

(2) 使用属性声明语法对公共属性进行的定义。该语法通过 get 和 set 访问函数将专用数据成员和公共属性关联起来。

在定义属性时,不能申明为 void 类型,即属性必须具有一个名称和一个类型。set 函数常使用关键字 value,value 的类型必须同它被分配到的属性的声明类型相同。

下面的程序为类定义了一个 Name 属性。

```
private string m_Name;
public string Name
{
    get
    {
        return m_Name;
    }
    set
    {
        m_Name=value;
    }
}
```

在属性定义中,通常包含专用数据成员,但这不是必需的。get 访问器不用访问私有数据成员就可以返回值。

3.5.3 定义索引器

索引器允许类或结构的实例按照与数组相同的方式进行索引。索引器类似于属性,和前述属性的区别在于索引器可以带有参数。this 关键字用于定义索引器。

下面是一个使用索引器的例子:

```
class IntArr
{
```

```
        private const int Length=6;
        private int[] buf=new int[Length];
        public int this[int index]
        {
            get
            {
                if (index<0 || index>=Length)
                {
                    System.ApplicationException ex=
                        new System.ApplicationException("数组越界。");
                    throw ex;
                }
                else
                {
                    return buf[index];
                }
            }
            set
            {
                if (index<0 || index>=Length)
                {
                    System.ApplicationException ex=
                        new System.ApplicationException("数组越界。");
                    throw ex;
                }
                else
                {
                    buf[index]=value;
                }
            }
        }
    }

    class Program
    {
        static void Main()
        {
            IntArr ia=new IntArr();
            ia[3]=3;
            ia[5]=5;
            //ia[9]=9;                       //引发异常
            //for (int i=0; i<=10; i++)      //引发异常
            for (int i=0; i<=5; i++)
            {
```

```
                System.Console.WriteLine("ia[{0}]={1}",i,ia[i]);
            }
        }
    }
```

程序运行结果为：

```
ia[0]=0
ia[1]=0
ia[2]=0
ia[3]=3
ia[4]=0
ia[5]=5
```

如果将程序中被注释掉的语句恢复，再次执行时将会引发异常。

说明：类 IntArr 中定义了一个整数数组 buf 来保存数据，并定义了索引器来访问 buf，如果索引超出数组范围，则抛出异常。

本例中索引器的参数为数组的下标。这只是使用索引器的一种情况，其实索引器不必根据整数值进行索引，完全可以由用户自行定义特殊的查找机制。

3.5.4 方法重载

下面用一个例子介绍方法的重载：

```
public class Student
{
    public string m_Mess;
    public virtual void SetMess()
    {
        m_Mess="You are a student.";
    }
}

public class GoodStudent:Student
{
    public override void SetMess()
    {
        m_Mess="You are a GOOD student.";
    }
}

class Program
{
    static void Main(string[] args)
```

```
    {
        Student student=new Student();
        GoodStudent goodstudent=new GoodStudent();

        student.SetMess();
        goodstudent.SetMess();
        Console.WriteLine(student.m_Mess);
        Console.WriteLine(goodstudent.m_Mess);
    }
}
```

程序中定义了两个类：Student 和 GoodStudent，GoodStudent 从 Student 继承。Student 中定义了方法 SetMess()，在 GoodStudent 中重新实现了该方法的定义。

运行结果为：

```
You are a student.
You are a GOOD student.
```

3.5.5 使用 ref 和 out 类型参数

ref 关键字使参数按引用方式传递。若要使用 ref 类型的参数，只能将变量作为 ref 参数显式地传递到方法。在调用方法之前，ref 参数必须先初始化。当控制传递回调用方法时，在方法中对参数所做的任何更改都将反映在该变量中。

与 ref 类似，out 关键字同样使参数按引用方式传递。不同之处在于：ref 参数要求变量必须在传递之前进行初始化，而 out 参数则不必；但参数 out 必须在函数内且函数结束之前，即传出值之前初始化，而 ref 参数则不必。请看下面示例：

```
static void f1(ref int i)
{
    i=1;                    //没有这句也不会编译出错
}
static void f2(out int i)
{
    i=2;                    //没有这句则会编译出错
}

static void Main()
{
    int val1=0;             //如果不赋初值,则会编译出错
    f1(ref val1);
    Console.WriteLine(val1);

    int val2;               //可以不赋初值
    f2(out val2);
```

```
    Console.WriteLine(val2);
}
```

运行结果为:

```
1
2
```

3.5.6 抽象类和接口

抽象类是用 abstract 修饰符修饰的类,表示该类是不完整的,它只能用做基类。抽象类与非抽象类在以下方面是不同的。

(1) 抽象类不能直接实例化,对抽象类使用 new 操作符会编译出错。虽然一些变量和值在编译时的类型可以是抽象的,但是这样的变量和值必须或者为 null,或者含有对非抽象类的实例的引用。

(2) 允许(但不要求)抽象类包含抽象成员。

(3) 抽象类不能被密封。

当从抽象类派生非抽象类时,这些非抽象类必须真正实现所继承的所有抽象成员,从而重写那些抽象成员,如:

```
abstract class A
{
    public abstract void F();
}
abstract class B: A
{
    public void G() {};
}
class C: B
{
    public override void F()
    {
        //真正实现 F
    }
}
```

抽象类 A 引入抽象方法 F。类 B 引入另一方法 G,但由于它不提供 F 的实现,B 也必须声明为抽象类。类 C 重写 F,并提供一个具体实现。由于 C 中没有了抽象成员,因此可以(但并非必须)声明为非抽象类。

一个接口定义一个协定。实现某接口的类或结构必须遵守该接口定义的协定,一个类或结构可以实现多个接口。

在 .NET Framework 中,接口可以包含方法、属性、事件和索引器。接口本身不提供它所定义的成员的实现。接口只指定实现该接口的类或结构必须提供的成员,实现接口

的任何类都必须提供接口中所声明的抽象成员的定义。

习　　题

1. C#语言控制台应用程序的入口在哪里?
2. 如何为C#语言程序增加注释?
3. C#语言支持哪些数据类型? 与C++语言相比有哪些特点?
4. C#语言的值类型和引用类型有何区别?
5. 在C#语言中结构类型和类的区别是什么?
6. C#语言引入装箱和拆箱概念有何意义?
7. 请简述装箱和拆箱的过程。
8. Console类都提供了哪些输入输出方法?
9. switch语句在C#语言与C语言中有哪些异同点?
10. 判断下列写法的正误,如果有错误请指出错误原因:

(1)

```
if (nValue=5)i=1;
```

(2)

```
int[] nValue={1,2,3,4,5};
foreach(int n  in  nValue)
{
    n++;
    Console.WriteLine(n);
}
```

11. 错误和异常有什么区别? 为什么要进行异常处理? 用于异常处理的语句有哪些?
12. 编写一个程序段,输出1~5的平方值,要求:

(1) 用for语句实现;
(2) 用while语句实现;
(3) 用do-while语句实现。

13. 编写一个程序段,输出Fibonacci数列的前十位数值。
14. 编写一个程序段,接收一个长度大于4的字符串,并完成下列功能:

(1) 输出字符串的长度;
(2) 输出字符串中第一次出现字母a的位置;
(3) 在字符串的第4个字符后面插入子串"hello",输出新字符串;
(4) 将字符串"hello"替换为"world",输出新字符串;
(5) 以第3个字符为分隔符,将字符串分离,并输出分离后的字符串。

15. 请简要说明抽象类和接口的主要区别。

16. 编写一段程序代码,完成下列功能,并回答提出的问题。

(1) 创建一个类 ClassA,在构造函数中输出"A",再创建一个类 ClassB,在构造函数中输出"B"。

(2) 创建一个新类 ClassC 继承自类 ClassA,在 ClassC 内创建一个成员 B。不要为 ClassC 创建构造函数。

(3) 创建类 MainClass,在 Main 方法中创建类 ClassC 的一个对象,写出运行程序后输出的结果。

(4) 如果在 ClassC 中也创建一个构造函数输出"C",整个程序运行的结果又是什么?

第 4 章 chapter 4

ASP.NET 基本控件

Web 应用程序的开发，其核心是生成呈现给用户的 Web 页面，而呈现给用户的 Web 页面总是由各类控件构成的，因此，从本章开始，我们首先介绍构成 Web 页面的各类控件的属性及编程控制方法。

4.1 控件概述

在介绍具体的控件之前，本节先介绍有关 ASP.NET 控件的一些基本概念。

4.1.1 Web 控件的分类

我们前面已经接触过一些控件，如 HTML 的按钮、输入框等，用户通过各类控件与应用系统交互。ASP.NET 加强了控件的功能，特别是增强了控件在服务器端的处理能力。对 ASP.NET 来说，Web 控件共包括以下四种类型。

1. HTML 控件

最初可用于任何 HTML 页面的控件，也都可用在 ASP.NET 页面中，这部分内容在 1.1 节中已经介绍，这里不再赘述。

2. HTML 服务器控件

HTML 服务器控件在 HTML 控件的基础上加以改进，其功能有所增强，最重要的是可以在服务器端进行处理。

原始的 HTML 控件，如<input>等，在服务器端不做处理，直接发送到客户端浏览器进行显示。只要简单地为 HTML 控件加上 runat="server"属性设置，即可将其转换为 HTML 服务器控件；再为其加上 id 等属性，即可在服务器端对其进行编程处理。

ASP.NET 支持 HTML 服务器控件的一个重要原因是：在 ASP.NET 环境下对已有的 HTML 页面进行方便的改进。例如，对于一个已有的 input 控件<input type="text">，将其改为<input type="text" id="BookTitle" runat="server">，就变成了 HTML 服务器控件，就可以在服务器端通过其 id 对其内容进行存取了。

3. ASP. NET 服务器控件

也称为 ASP 控件，是 ASP. NET 的核心内容之一。它们在服务器端集成，遵循 . NET Framework 面向对象的编程模型。ASP. NET 服务器控件执行时在客户端表现为 HTML，但具有更强的服务器端处理能力，从而在大多数情况下替代了传统的 HTML 控件。除了功能更强、种类更多之外，ASP. NET 服务器控件克服了传统 HTML 控件在属性设置方面的缺点，可在服务器端通过程序预置。

4. 用户自定义控件

由开发人员自行创建的控件。在本书的应用实例中，大量用到了用户自定义控件。对用户自定义控件的详细介绍见 10.3 节。

本章主要介绍 ASP. NET 服务器控件。

ASP. NET 服务器控件可以在页面内容文件中直接拖动、配置（设计期），也可以使用 C# 语言等任何 . NET Framework 支持的语言在源程序中创建并处理（运行期）。与传统的客户端 HTML 控件相比，ASP. NET 控件有如下改进。

（1）自动检测浏览器版本，对支持 DHTML 的浏览器，则将脚本一起发送到客户端处理；对于低版本的浏览器，则将所有处理都回送到服务器端进行。

（2）使用编译型语言替代解释型脚本语言，目的是获得更好的性能。

（3）能够与数据源绑定。

（4）事件在客户端浏览器上触发，在服务器端处理。

（5）每个 ASP. NET 控件都由类实现。

这些优点在此讨论还显得有些空洞，随着后面章节的介绍，读者肯定会对这些优点产生确实而深入的理解。

ASP. NET 使用事件驱动模式进行处理。ASP. NET 服务器控件是可以触发事件的对象。用户在浏览器上对 ASP. NET 控件所执行的任何行为都可能触发事件；服务器端代码响应事件，并运行事件处理方法中的代码。

所有的 ASP. NET 事件都在服务器端处理，这与传统 HTML 控件事件的处理方式有本质区别（ASP. NET 控件都有一个 runat＝server 属性）。有些事件触发后立即发送到服务器，另一些事件则在触发后被存储，直到下一次页面回传到服务器时再处理。

4.1.2 ASP.NET服务器控件常用的属性和事件

编程使用控件时，主要工作是对控件的属性和事件进行处理。所有呈现在浏览器的、具有可视化外观的 ASP. NET 服务器控件都从 WebControl 类（位于 System. Web. UI. WebControls 命名空间）派生。该类提供了所有 ASP. NET 服务器控件的通用属性、方法和事件。

表 4-1 给出了 WebControl 类常用的属性和事件，所有 ASP. NET 服务器控件都会继承这些属性和事件。这样，就不必在后面介绍每个控件时再重复介绍了，而将精力集中在各控件特有的性质上。

表 4-1　所有 ASP.NET 服务器控件常用的属性和事件

属性名称	说　　明
AccessKey	快捷键
BackColor	控件的背景色
BorderColor	控件的边框颜色
BorderStyle	控件的边框样式
BorderWidth	控件的边框宽度
Controls	当前对象所包含的所有子控件
CssClass	控件所使用的级联样式表(CSS)类
Enabled	控件是否可用
EnableTheming	是否对此控件应用主题
EnableViewState	指示服务器控件是否向发出请求的客户端保持自己的视图状态以及它所包含的任何子控件的视图状态
Font	控件的字体属性
ForeColor	控件的前景色(通常是文本颜色)
Height	控件的高度
ID	控件的编程标识符
Parent	对该控件父控件的引用
SkinID	应用于当前控件的外观
ToolTip	当鼠标指针悬停在当前控件上时所显示的文本
Visible	控件是否可见
Width	控件的宽度
事件名称	**说　　明**
DataBinding	当控件绑定到数据源时发生
Disposed	当控件从内存中被释放时发生，这是控件生存期的最后阶段
Init	当控件被初始化时发生，这是控件生存期的第一步
Load	当控件被加载到页面时发生
PreRender	在控件加载之后、呈现之前发生
Unload	当控件从内存中卸载时发生

下面对控件的上述属性做个简要介绍，不可能涉及所有细节，详细的使用请查阅 MSDN 中关于 WebControl 类的介绍。

AccessKey 属性的值为一个字母，在程序的运行中，同时按下 Alt 键和此键可将输入焦点移到此控件。

BorderStyle 属性设置控件的边框样式，其值为一个 BorderStyle 枚举值，包括：

- NotSet：不设置边框样式(默认值)；
- None：无边框；
- Dotted：虚线边框；
- Dashed：点划线边框；
- Solid：实线边框；
- Double：双实线边框；
- Groove：凹槽状边框；
- Ridge：突起边框；
- Inset：内嵌边框；
- Outset：外嵌边框。

BorderWidth 设置控件的边框宽度。如果其值为一个整数，则其单位为像素，如果其值为整数加“px”，则其单位为点。

有些 ASP. NET 服务器控件是容器控件，在其中可能包含多个子控件，这些子控件可通过一个 ControlCollection 对象类型的 Controls 属性来获取。例如，要移除控件 myControl 的所有子控件，可使用如下代码：

```
myControl.Clear();
```

用下面代码也可以达到同样的效果：

```
for (int i=myControl.Controls.Count-1; i>=0; i--) {
    myControl.Controls.Remove(myControl.Controls[i]);
}
```

EnableTheming 属性指示当前控件是否使用主题。当该属性为 True 时，将在应用程序的主题目录中搜索要使用的控件外观；当该属性为 False 时，则不会搜索主题目录，并且不会使用 SkinID 属性的内容。有关主题的内容本书并未涉及，有兴趣的读者请参阅相关资料。

EnableViewState 属性的默认值为 True。所谓“视图状态”是指服务器控件的所有属性值的集合。当 EnableViewState 属性的值为 True 时，ASP. NET 可在 HTTP 请求之间维持视图状态，但有时为了提高性能，可将其值设为 False。

Font 属性包含多个子属性，如 Bold、Italic、Name、Names、Strikeout、Underline 和 Size 等。这些子属性在页面中声明时可使用 Property-Subproperty 形式(如 Font-Size="10pt")进行访问，而编程时则可以使用 Property. Subproperty 形式(如 Label1. Font. Underline=True)进行访问。

Height 和 Width 属性表示控件的高度和宽度，也有两种形式。如果其值为一个整数，则其单位为像素，如果其值为整数加“%”，则表示其高度和宽度在容器中所占的百分比。

ID 是控件的编程标识符。如果事先对此属性进行了设置，则可在程序中通过 ID 对控件的属性、事件和方法进行访问。如果没有事先指定该属性，也可通过其父控件的

Controls 属性获取对该控件的引用。

4.1.3 事件驱动与事件处理

ASP. NET 的一个核心概念就是事件驱动。

多数情况下，事件由用户在操作控件时触发，由程序进行处理。

用户在浏览器上对服务器控件所执行的任何操作都可能触发事件。服务器端代码响应事件，并运行存储在事件处理方法中的代码。

ASP. NET 事件有数千个之多，所有 ASP. NET 事件都在服务器端进行处理。有些事件立刻发送到服务器，另外有一些事件则被存储，等到下次页面回传到服务器时再处理。

假设页面上有一个 ASP. NET Button 控件，它不同于以往的 HTML Button 控件，具有一个 runat="server"属性，表示处理是在服务器端进行。当用户用鼠标单击了该按钮，浏览器则捕获 1 个客户端事件，并将页面以 HTTP POST 方式传回服务器。服务器判断有无与该单击事件相关的事件处理程序，如有，则执行之。上述过程是自动进行的，程序员的工作只是为特定的事件编写处理程序。

并不是所有的客户端事件都引起页面回传，如果像 MouseOver 这样的事件也将页面回传到服务器端进行处理的话，那程序的运行效率也太低了。

如果按客户端事件是否需要回传到服务器进行处理来划分控件，ASP. NET 控件可划分为以下两类。

一些控件本身就代表对一种操作的选择，主要用于处理单击事件，如 Button 控件、Calendar 控件、DataGrid 控件、DataList 控件、FileUpload 控件、GridView 控件、ImageButton 控件、ImageMap 控件、LinkButton 控件、Menu 控件、Repeater 控件等。当用户单击这些控件上的交互部分（如这些控件上的按钮、超链等）时，总会引起页面的回传。

还有一些控件一般用于对数据的操作或选择。操作数据往往需要一个过程，也往往是在操作最终确认后才需要回传服务器进行处理。这类控件包括 CheckBox 控件、CheckBoxList 控件、DropDownList 控件、ListBox 控件、RadioButtonList 控件、RadioButton 控件、TextBox 控件等。这类控件都包含一个 AutoPostBack 属性，默认情况下该属性值为 False。当 AutoPostBack 属性值为 False 时，即使这些控件中的内容被修改了，也不会将页面回送服务器；而是等到有其他控件触发了需要回送服务器的事件后，这些控件的事件处理程序才会被“顺带”地执行。但如果将这类控件的 AutoPostBack 属性改为 True，当控件内容改变时，也会自动将页面回送到服务器进行处理。

一般情况下，ASP. NET 事件的处理函数都会有两个参数，并且无返回值。

第一个参数表示触发当前事件的对象，按惯例被称为 sender。大多数情况下是不需要操作此参数的，但作为通用的事件处理函数，传递此参数是相当必要的。

第二个参数被称为事件参数，包含了与事件相关的特殊信息。其实大多数事件处理函数并不需要传递事件信息，对于这些事件，此参数类型为 EventArgs。EventArgs 不包括任何属性，表示不包含任何事件信息。这些事件处理函数的原型为：

```
private void EventName (object sender,EventArgs e)
```

对于那些包含事件信息的事件,其处理函数的第二个参数是从 EventArgs 派生的类型。

例如:Button 控件的 Command 事件处理函数,其原型为:

```
protected void CommandBtn_Click(object sender,CommandEventArgs e)
```

其第二个参数为 CommandEventArgs 类型,包含 CommandName 和 CommandArgument 等属性,详见 4.2.2 节。

4.2 一般控件

学习使用 ASP.NET 控件的最好方法就是边学习边在实际的页面中进行实践,本节就结合实例介绍 ASP.NET 最简单、常用的控件。

4.2.1 Label 控件

其实,在 2.2 节的“第一个应用程序”中就使用了 Label 控件。Label 控件用来显示用户不能编辑的静态文件,它本身的 Text 属性包含了要显示的文本字符串。

创建一个名为 UseLabel 的网站,并在其默认的主页 Default.aspx 上进行操作。创建网站的方法可参考 2.2 节的相关内容。

拖动一个 Label 控件到页面上来,原始的代码如下:

```
<asp:Label ID="Label1" runat="server" Text="Label"></asp:Label>
```

提示:可在源视图中直接修改代码来改变其属性;也可以在设计视图中先选中 Label 控件,然后在集成开发环境右下部的属性设置窗口中设置相关属性。

例如,可将 Label 控件的 Text 属性设置为“这是一个 Label 控件”,将 Font-Bold(粗体)属性设置为 True,将 Font-Names(字体)属性设置为“楷体_GB2312”,将 ForeColor(文字颜色)属性设置为 Blue。修改后的代码如下:

```
<asp:Label ID="Label1" runat="server" Text="这是一个 Label 控件。"
    Font-Bold="True" Font-Names="楷体_GB2312" ForeColor="Blue">
</asp:Label>
```

在浏览器中查看该页面,效果如图 4-1 所示。

除了直接在页面上修改 Label 控件的属性外,还可以通过编程来对其属性进行修改。

在第一个 Label 控件的下面再为页面增加一个 Label 控件,其 ID 为 Label2。在代码编辑窗口中单击鼠标右键,在弹出菜单中选择“查看代码”项;或在右侧的解决方案资源管理器中展开 Default.aspx 后双击打开 Default.aspx.cs 文件,可对当前页面的程序代码进行编辑(有关代码隐藏的内容请参考本书 6.1 节)。

代码中已经有了一个 Page_Load()函数,为其增加如下代码:



图 4-1　Label 控件的执行效果(一)

这是一个Label控件。
2007-11-19 21:23:47

图 4-2　Label 控件的执行效果(二)

```
if (!Page.IsPostBack)
{
    Label2.Text= DateTime.Now.ToString();
}
```

再次执行页面,结果如图 4-2 所示。

提示: 使用了语句 if (!Page.IsPostBack),是确保页面只在首次加载时才执行相关代码;在以后的回送中则不再执行,而是从“视图状态”中自动重新获取数据,详见 4.1 节中对 EnableViewState 属性的说明。

编程控制所要显示的文本是使用 Label 控件的根本原因,如果文本内容不需要修改,还是应该直接使用传统的 HTML 文本,以减少服务器端的处理。

4.2.2　Button 控件

Button 控件是最常用的交互控件之一,一般情况下,用户在客户端单击 Button 控件后,就会将表单提交到服务器进行处理。

Button 控件所特有的属性和事件如表 4-2 所示。

表 4-2　Button 控件所特有的属性和事件

属性名称	说　明
CommandName	Button 控件命令名,该命令名可传递给 Button 控件的 Command 事件,并在事件处理函数中进行相应的处理
CommandArgument	命令可选参数,该参数与 CommandName 一起被传递到 Command 事件
事件名称	**说　明**
Click	在单击 Button 控件时发生
Command	在单击 Button 控件时发生

创建一个名为 UseButton 的网站。

参照 4.2.1 小节的方法,为网页增加一个 Label 控件,其 ID 为 Label1。同样将其 Text 属性设置为“这是一个 Label 控件”,将 Font-Bold(粗体)属性设置为 True,将 Font-Names(字体)属性设置为“楷体_GB2312”,将 ForeColor(文字颜色)属性设置为 Blue。

在 Label 控件下面再增加一个 Button 控件,改变其 Text 属性,在设计视图中双击该

Button 控件,系统会自动转到源代码文件的编辑,并自动为 Button 控件创建 Click 事件处理函数 Button1_Click()。

页面代码中修改后的 Button 控件相关部分如下:

```
<asp:Button ID="Button1" runat="server" Text="改变 Label 的显示"
    OnClick="Button1_Click" />
```

在函数 Button1_Click()中增加如下代码:

```
Label1.Text="Label 上的文本被 Button 所修改";
```

在浏览器中查看该页面,Label 控件的初始显示为“这是一个 Label 控件”。单击“改变 Label 的显示”按钮,可以看到页面被重新加载,Label 控件的显示改为“Label 上的文本被 Button 所修改”。

从表 4-2 可以看出,当单击 Button 控件时既可以触发 Click 事件,也可以触发 Command 事件。当为 Button 控件指定了命令名时,通常使用 Command 事件进行处理。

如果一个网页上有多个 Button 控件,它们有大量相似的处理操作,为每个 Button 控件分别编制单击事件处理函数比较麻烦。这时,一个好的选择就是:为每个 Button 控件指定不同的命令名和相同的 Command 事件处理函数,并在 Command 事件处理函数中判断单击的是哪个 Button 控件并进行相应的处理。

为页面再增加 4 个按钮,改变其属性后代码如下:

```
<asp:Button ID="Button2" runat="server" CommandName="FontBold"
    OnCommand="CommandBtn_Click" Text="变粗体" /><br />
<asp:Button ID="Button3" runat="server" CommandName="FontUnderline"
    OnCommand="CommandBtn_Click" Text="加下划线" /><br />
<asp:Button ID="Button4" runat="server" CommandArgument="Red"
    CommandName="FontColor" OnCommand="CommandBtn_Click" Text="变为红色" />
<asp:Button ID="Button5" runat="server" CommandArgument="Green"
    CommandName="FontColor" OnCommand="CommandBtn_Click" Text="变为绿色" />
```

提示:改变控件的属性可在源视图中直接修改代码来完成;也可以在设计视图中先选中控件,然后在集成开发环境右下部的属性设置窗口中设置相关属性。如果想在属性设置窗口中为按钮指定 Command 事件处理函数,需要先在属性设置窗口中单击“闪电”图标,属性设置窗口中会列出当前控件的所有事件,在 Command 中输入函数名即可。双击该项即可直接进入该函数的程序代码编辑界面。

从上述代码可以看出,4 个按钮共用相同的 Command 事件处理函数,其代码如下:

```
protected void CommandBtn_Click(object sender,CommandEventArgs e)
{
    switch (e.CommandName)
    {
        case "FontBold":
            Label2.Font.Bold=True;
```

```
            break;
        case "FontUnderline":
            Label2.Font.Underline=True;
            break;
        case "FontColor":
            if ((String)e.CommandArgument=="Red")
                Label2.ForeColor=Color.Red;
            if ((String)e.CommandArgument=="Green")
                Label2.ForeColor=Color.Green;
            break;
        default:
            break;
    }
}
```

函数的第二个参数是从 EventArgs 派生的 CommandEventArgs 类型，包含 CommandName 和 CommandArgument 等属性，以携带从控件传来的事件信息。

虽然此函数被多个 Button 控件所共用，但不同控件的 CommandName 属性各不相同，因此，程序可根据此属性对不同的按钮做出不同的响应。如果需要的话，还可以进一步根据 CommandArgument 属性进行区分处理。

代码中使用了一个 switch 语句，根据不同的 CommandName 进行不同的处理。如果 CommandName 为 FontColor，则还需要进一步对 CommandArgument 属性进行判断。

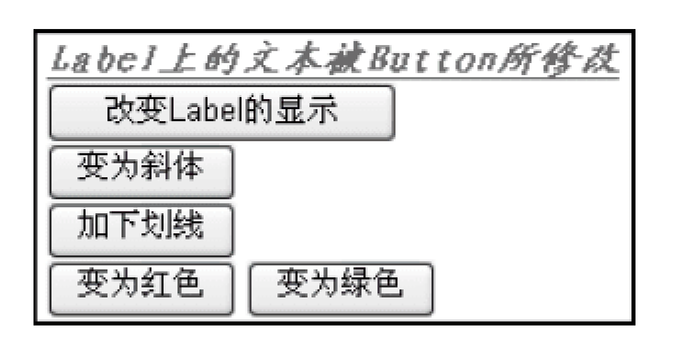


图 4-3　Button 控件的执行效果

提示：*要使用 Color 结构所预定义的颜色值，需要在程序代码的前部加上"using System.Drawing;"语句。*

页面的执行效果如图 4-3 所示。

另外，与 Button 控件类似的控件还有 LinkButton 和 ImageButton。它们的功能都与 Button 控件相同，都可以对单击事件做出响应，但 LinkButton 控件的外观为一个超链，而 ImageButton 的外观为一个图片。

4.2.3　TextBox 控件

TextBox 控件用于接受用户的输入，也是最常用的交互控件之一，它也可以用于显示只读文本。可以把它配置为单行或多行模式，还可以配置为接受密码。如果设置为多行，那么它将自动换行，除非 Wrap 属性被设置为 False。

TextBox 控件所特有的属性和事件如表 4-3 所示。

默认情况下 TextBox 控件中的内容被修改后不会将页面回送服务器。但如果将 TextBox 控件的 AutoPostBack 属性改为 True，当 TextBox 控件失去输入焦点时，如果其内容已改变，则会将页面回送到服务器进行处理。有关 AutoPostBack 属性更详细的说明请参考 4.1 节。

表 4-3 TextBox 控件所特有的属性和事件

属性名称	说明
AutoPostBack	当用户在 TextBox 控件中按 Enter 或 Tab 键时，TextBox 控件将失去输入焦点时。本属性确定当控件失去输入焦点时，页面是否自动回传到服务器进行处理，默认为 False
Columns	文本框的显示宽度(以字符为单位)
MaxLength	文本框中最多允许输入的字符数
ReadOnly	TextBox 控件的内容是否只读，默认为 False
Rows	多行文本框中显示的行数
Text	文本内容
TextMode	控件的行为模式(单行、多行或密码)
ValidationGroup	当前控件回发到服务器时导致验证的控件组
Wrap	多行文本框内的文本内容是否自动换行
事件名称	**说明**
TextChanged	当向服务器发送时，如果文本框的内容已更改，则触发此事件

TextMode 属性确定控件的行为模式。TextBox 控件的行为模式如下。

- MultiLine：表示多行输入模式；
- Password：表示密码输入模式；
- SingleLine：表示单行输入模式(默认值)。

当 TextBox 控件处于多行模式时，可以通过设置 Rows 属性来控制显示的行数。还可以通过设置 Wrap 属性来指定文本是否自动换行。如果 TextBox 控件处于密码模式，则会屏蔽该控件中输入的所有字符。

创建一个名为 UseTextBox 的网站。

在页面上增加两段文本和两个 TextBox 控件。将第一个 TextBox 控件的 AutoPostBack 属性改为 True，在设计视图中双击该 TextBox 控件，系统会自动转到源代码文件的编辑，并自动为 TextBox 控件创建 TextChanged 事件处理函数 TextBox1_TextChanged()。

页面相关部分代码如下：

```
TextBox1：<asp：TextBox ID="TextBox1" runat="server" AutoPostBack="True"
    OnTextChanged="TextBox1_TextChanged"></asp：TextBox>
TextBox2：<asp：TextBox ID="TextBox2" runat="server"></asp：TextBox>
```

在函数 TextBox1_TextChanged()中增加如下代码：

```
TextBox2.Text=TextBox1.Text;
```

在浏览器中查看该页面，效果如图 4-4 所示。

开始时两个 TextBox 控件的内容都为空，在第一个 TextBox 控件中输入一段文本，

TextBox1: Hello World!　TextBox2: Hello World!

图 4-4　TextBox 控件的执行效果

当该控件失去输入焦点时(如在页面上该控件以外的其他地方单击鼠标),可以看到页面被重新加载,TextBox2 的内容被置为与 TextBox1 相同。

注意:即使将 TextBox 控件的 AutoPostBack 属性设为 True,也不是在文本框中每输入一个字符就引起回传,而是只有当文本框失去输入焦点并且文本框内的文本已经改变才会引起回传。

TextBox 控件在浏览器中可见,还可以对它进行编程控制。在编程上与其相似的控件有 HiddenField 控件,但 HiddenField 控件在界面上是不可见的。当开发者想要处理页面上的信息,又不想让用户看到这些信息时,HiddenField 控件是一个很常用的选择。HiddenField 控件的使用与 HTML 控件的＜input type="hidden" value=""＞相似,这里就不再详细介绍了。

4.2.4　HyperLink 控件

在传统的 HTML 页面上使用＜a href="" Target=""＞＜/a＞标记建立超链。在 ASP.NET 中,可以使用 HyperLink 控件,它与 HTML 控件的执行效果相同,但在服务器端可以得到更好的编程处理能力。

与 Button 控件不同,在客户端单击 HyperLink 控件后不向服务器回送页面,而是直接导航到目标 URL。

HyperLink 控件所特有的属性如表 4-4 所示。

表 4-4　HyperLink 控件所特有的属性

属性名称	说　明
ImageUrl	为 HyperLink 控件所显示的图像的路径
NavigateUrl	单击 HyperLink 控件时链接到的 URL
Target	单击 HyperLink 控件时显示链接到网页内容的目标窗口或框架
Text	HyperLink 控件的文本标题

ImageUr 属性值为要显示的图像的路径。正常情况下,HyperLink 控件在页面上显示为文字的超链;但如果设置了 ImageUrl 属性,HyperLink 控件则显示为一个图像。

Text 属性为显示在浏览器中的链接文本。如果同时设置了 Text 属性和 ImageUrl 属性,则 ImageUrl 属性优先,只有当图片无效时,才显示文本内容。

默认情况下,单击 HyperLink 控件时,目标网页加载到当前浏览器窗口或当前框架(如果使用了框架,有关框架的内容请参考 1.1.2 小节)内,但可以通过改变 Target 属性来改变加载内容的目标窗口或框架。

Target 属性的值可以是以下特殊值之一。

- _blank:将内容呈现在一个没有框架的新窗口中;

- _parent：将内容呈现在当前框架的父框架中，如果没有父框架，则此值等同于_self；
- _search：在搜索窗格中呈现内容；
- _self：将内容呈现在当前框架中(默认值)；
- _top：将内容呈现在当前整个窗口中(忽略原有的框架)。

除以上特殊值外，Target 属性的值还可以是目标框架的名称。

创建一个名为 UseHyperLink 的网站。

从工具箱中将一个 HyperLink 控件拖到页面上，修改其属性。再建立一个传统超链与其相比较。相关页面代码如下：

```
<asp:HyperLink ID="HyperLink1" runat="server" NavigateUrl="DestPage.aspx" Target="_blank">
HyperLink控件</asp:HyperLink>
<br />
<a href="DestPage.aspx" Target="_blank">效果相同的传统超链</a>
```

执行时页面上会出现两个超链，分别单击它们，所得到的效果完全相同，都是新打开一个浏览器窗口显示目的页面。

提示：要正确运行上述网站，还需要创建一个称为 DestPage.aspx 的页面作为目的页面。

4.3 选择控件

应用程序经常会请求用户做出各种各样的选择，因此，ASP.NET 提供了多种选择控件。本节主要介绍其中最常用的 CheckBox 控件、RadioButton 控件、ListBox 控件和 DropDownList 控件。

4.3.1 CheckBox 控件

当允许用户在有限种可能性中选择多个时，使用复选框。

CheckBox 控件用来显示一个复选项，供用户进行 True 或 False 的选择，CheckBox 控件所特有的属性和事件如表 4-5 所示。

表 4-5 CheckBox 控件所特有的属性和事件

属性名称	说明
AutoPostBack	当用户单击 CheckBox 控件而改变了它的选中状态时，是否自动回发到服务器
Checked	是否已选中 CheckBox 控件
Text	显示在网页上的 CheckBox 文本标签
TextAlign	文本标签的对齐方式
事件名称	**说明**
CheckedChanged	当向服务器回送页面时，如果 Checked 属性的值已更改，则触发此事件

与 TextBox 控件相同，默认情况下，改变 CheckBox 的选择不会引发页面回送服务器。但如果将 CheckBox 控件的 AutoPostBack 属性改为 True，当改变 CheckBox 的选择后，则会将页面回送到服务器进行处理。

Checked 属性说明控件的选中状态，当其值为 True 时，说明该控件已被选中。

TextAlign 属性的值可以是以下两个枚举值之一。

- Left：文本标签显示在控件的左侧；
- Righ：文本标签显示在控件的右侧。

创建一个名为 UseCheckBox 的网站。

在页面上增加一个 Label 控件和两个 CheckBox 控件。分别将两个 CheckBox 控件的 AutoPostBack 属性改为 True；将第二个 CheckBox 控件的 TextAlign 属性的值改为 Left；在设计视图中分别双击该两个 CheckBox 控件，系统会自动转到源代码文件的编辑，并自动为其创建 CheckedChanged 事件处理函数。

相关的页面代码如下：

```
<asp:Label ID="Label1" runat="server" Text="这是一个 Label 控件。"
Font-Bold="True" Font-Names="楷体_GB2312" ForeColor="Blue"></asp:Label>
<br />
<asp:CheckBox ID="CheckBox1" runat="server" Text="斜体"
    AutoPostBack="True" OnCheckedChanged="CheckBox1_CheckedChanged" />
<asp:CheckBox ID="CheckBox2" runat="server" Text="下划线"
    AutoPostBack="True" OnCheckedChanged="CheckBox2_CheckedChanged"
    TextAlign="Left" />
```

两个 CheckBox 控件的 CheckedChanged 事件处理函数代码如下：

```
protected void CheckBox1_CheckedChanged(object sender,EventArgs e)
{
    if (CheckBox1.Checked)
        Label1.Font.Italic=True;
    else
        Label1.Font.Italic=False;
}
```

上述代码根据 CheckBox1 当前是否被选中的状态，改变控件 Label1 字体的“斜体”属性。

```
protected void CheckBox2_CheckedChanged(object sender,EventArgs e)
{
    if (CheckBox2.Checked)
        Label1.Font.Underline=True;
    else
        Label1.Font.Underline=False;
}
```

上述代码根据 CheckBox1 当前是否被选中的状态，改变控件 Label1 字体的“下划

线"属性。

在浏览器中查看该页面,效果如图4-5所示。

在本例中可以看出,各CheckBox控件之间是相互独立的。即使排列在一起供多项选择,也是每个CheckBox控件有自己独立的属性设置和处理事件(当然,可以为多个CheckBox控件的事件设定相同的事件处理函数)。



图4-5 CheckBox控件的执行效果

在外观上与CheckBox控件相同的ASP.NET控件还有CheckBoxList,它是一个多项选择复选框组。该复选框组可以通过编程的方式动态创建,也可绑定到数据源动态创建。CheckBoxList控件有统一的属性设置和事件处理。

4.3.2 RadioButton控件

当只允许用户在有限种可能性中选择一种时,使用单选按钮。

RadioButton控件用来显示一个单选按钮。整个页面上的所有RadioButton控件按照GroupName属性进行分组,GroupName属性相同的为一组,同一组中同时只能有一个RadioButton控件可以被选中。

RadioButton控件与CheckBox控件很相似,具有相似的属性和事件,只是多了一个GroupName属性。RadioButton控件与CheckBox控件不同的属性如表4-6所示。

表4-6 RadioButton控件与CheckBox控件不同的属性

属性名称	说明
GroupName	单选按钮所属的组名

创建一个名为UseRadioButton的网站。

在页面上增加两组RadioButton控件,其GroupName属性分别为ColorGroup和FontGroup。ColorGroup组包括两个RadioButton控件,FontGroup组包括三个RadioButton控件,相关的页面代码如下:

```
颜色:<br>
<asp:RadioButton ID="RadioButton1" runat="server" GroupName="ColorGroup" Text="红色" />
<asp:RadioButton ID="RadioButton2" runat="server" GroupName="ColorGroup" Text="蓝色" />
<br>字体:<br>
<asp:RadioButton ID="RadioButton3" runat="server" GroupName="FontGroup" Text="宋体" />
<asp:RadioButton ID="RadioButton4" runat="server" GroupName="FontGroup" Text="黑体" />
<asp:RadioButton ID="RadioButton5" runat="server" GroupName="FontGroup" Text="楷体" />
```

页面的执行效果如图 4-6 所示。

颜色：
○红色 ◉蓝色
字体：
◉宋体 ○黑体 ○楷体

图 4-6　RadioButton 控件的执行效果

页面上的 5 个 RadioButton 控件分为两组。在颜色组中只能选择一个控件，当用户选中其中一个时，另一个则自动取消选中。在字体组中同样也是只能选择一个。

本示例只演示了 RadioButton 控件的界面特征，针对 RadioButton 的编程与 CheckBox 控件非常相似，就不再介绍了。

ASP.NET 也提供了一个将多个 RadioButton 控件封装在一起形成的 RadioButtonList 控件。

4.3.3　ListBox 控件

ListBox 控件是一种列表选择控件。它将所有可选项列在列表框内，如果可选项太多，列表框会出现滚动条。ListBox 控件所特有的属性和事件如表 4-7 所示。

表 4-7　ListBox 控件所特有的属性和事件

属性名称	说　　明
Items	列表项的集合
Rows	控件中显示的行数
SelectedIndex	列表中选定项的最低序号索引
SelectedItem	ListBox 控件中索引最小的选定项
SelectedValue	ListBox 控件中选定项的值
SelectionMode	ListBox 控件的选择模式
Text	与 SelectedValue 属性的值相同
事件名称	**说　　明**
SelectedIndexChanged	当 ListBox 控件的选定项在信息发往服务器之间变化时发生
TextChanged	当 Text 和 SelectedValue 属性更改时发生

Items 属性的类型为 ListItemCollection 类。ListItemCollection 类不能被继承，其常用属性如下。

- Capacity：可以存储的最大项数。
- Count：集合中的 ListItem 对象数。
- IsReadOnly：是否为只读。
- Item：集合中指定索引处的 ListItem。

其常用公共方法如下。

- Add：将 ListItem 追加到集合的结尾。
- Clear：从集合中移除所有 ListItem 对象。
- Contains：确定集合是否包含指定的项。
- CopyTo：将 ListItemCollection 中的项复制到指定的 System.Array 中，从指定

的索引开始。

- Equals：确定两个 Object 实例是否相等。
- FindByText：搜索集合中具有 Text 属性且包含指定文本的 ListItem。
- FindByValue：搜索集合中具有 Value 属性且包含指定值的 ListItem。
- IndexOf：指定 ListItem 在集合中的位置。
- Insert：将 ListItem 插入集合中的指定索引位置。
- Remove：从集合中移除 ListItem。
- RemoveAt：从集合中移除指定索引位置的 ListItem。

可以通过上述属性和方法对 ListBox 控件中的项进行编程操作。

Items 当中的每一项是一个 ListItem 对象。ListItem 类包含以下常用属性。

- Enabled：是否启用当前列表项。
- Selected：当前项是否被选中。
- Text：当前项显示的文本。
- Value：当前项的值。

ListItem 类的构造函数有如下四种形式。

- ListItem()：初始化 ListItem 类的新实例。
- ListItem(String)：指定新实例的 Text 属性。
- ListItem(String,String)：指定新实例的 Text 和 Value 属性。
- ListItem(String,String,Boolean)：指定新实例的 Text、Value 和 Enabled 属性。

ListBox 控件中的项可单选也可多选。

通过将 SelectionMode 属性在 Single 和 Multiple 之间改变，可以控制 ListBox 是只能单选还是可以多选。单击某个选项可以选中它，多选的方法是：按住 Ctrl 键的同时分别在多个选项上单击鼠标；如果要选择连续的项，则可以在按住 Shift 键的同时分别单击第一个选项和最后一个选项，或直接在连续的选项上拖动鼠标。

如果 ListBox 控件只允许一个选项，则 SelectedIndex 属性可确定列表中当前选定项的索引。如果 ListBox 控件支持多个选项，则 SelectedIndex 属性可确定选定项的最小索引值。

同样，如果 ListBox 控件只允许一个选项，则使用 SelectedItem 属性可获取选定项的各个属性。如果 ListBox 控件允许多个选项，则使用 SelectedItem 属性可获取列表控件中索引最小的选定项的属性。

SelectedValue 属性返回当前选定项（ListItem 类型）的 Value 属性值。通常使用 SelectedValue 属性确定 ListBox 控件中选定项的值；如果选定了多个项，则返回索引最小的选定项的值。如果未选定任何项，则返回一个空字符串（""）。还可以通过对 SelectedValue 属性值的设定，使控件中具有该值的项被选定。如果 ListBox 控件中的任何项都不包含指定 SelectedValue 值，则会引发 System.ArgumentOutOfRangeException 异常。

创建一个名为 UseListBox 的网站。

在默认页 Default.aspx 上增加两段标题文本、两个 ListBox 控件和两个 Label 控件。

将两个 ListBox 控件的 AutoPostBack 属性改为 True，Rows 属性改为 5；将第二个 ListBox 控件的 SelectionMode 属性改为 Multiple。在设计视图中分别双击两个 ListBox 控件，系统会自动转到源代码文件的编辑，并自动为它们创建 SelectedIndexChanged 事件处理函数。

与 ListBox 控件相关的页面代码如下：

```
<h2>ListBox控件-单选：</h2>
<asp:ListBox ID="ListBox1" runat="server" AutoPostBack="True" Rows="5" OnSelectedIndexChanged="
ListBox1_SelectedIndexChanged"></asp:ListBox>
<br/>
<asp:Label ID="Label2" runat="server" Text="未选"></asp:Label>
<h2>ListBox控件-多选：</h2>
<asp:ListBox ID="ListBox2" runat="server" AutoPostBack="True"Rows="5" SelectionMode="Multiple"
OnSelectedIndexChanged="ListBox2_SelectedIndexChanged">
</asp:ListBox>
<br/>
<asp:Label ID="Label3" runat="server" Text="未选"></asp:Label>
```

在页面的 Page_Load()函数中增加如下代码：

```
if (!IsPostBack)      //如果页面是首次加载(不是回传的)
{
    //声明一个二维字符串数组,保存书名和编号
    string[,] books=
    {
        {"C语言程序设计","J01"},
        {"数据结构","J02"},
        {"网络程序设计","J03"},
        {".NET程序设计","J04"},
        {"大学英语","J05"},
        {"电子技术基础","J06"}
    };
    //将字符串数组的内容分别添加为两个 ListBox控件的选项
    for (int i=0; i<books.GetLength(0); i++)
    {
        ListBox1.Items.Add(new ListItem(books[i,0],books[i,1]));
        ListBox2.Items.Add(new ListItem(books[i,0],books[i,1]));
    }
}
```

说明：当页面第一次被加载时，程序中声明了一个二维字符串数组，保存书名和编号。循环处理，将数组的每一行作为一项分别加入到两个列表当中。

在程序中，调用 ListItem 的构造函数同时指定新实例的 Text 和 Value 属性，然后调用 Items 属性(类型为 ListItemCollection)的 Add 方法将新创建的 ListItem 对象增加到

列表中。

提示：ListBox控件的列表项不一定非要编程增加，也可以在设计期指定，其方法与4.3.4小节中对DropDownList控件的操作相同。

两个ListBox控件的SelectedIndexChanged事件处理函数代码如下。

```
protected void ListBox1_SelectedIndexChanged(object sender,EventArgs e)
{
    if (ListBox1.SelectedIndex != -1)              //如果有选中项
    {
        Label2.Text=ListBox1.SelectedItem.Value+"-"+
          ListBox1.SelectedItem.Text;
    }
}
```

第一个ListBox控件为单选列表。如果有选中项，则在Label2上同时显示选中项的值和显示文本。

```
protected void ListBox2_SelectedIndexChanged(object sender,EventArgs e)
{
    string str="";
    foreach (ListItem li in ListBox2.Items)        //检查ListBox2的每一项
    {
        if (li.Selected==True)                     //如果被选中
        {
            str+=li.Value+"-"+li.Text+"<br/>";
        }
    }
    //显示所有的选中项
    if (str.Length==0)
        Label3.Text="未选";
    else
        Label3.Text=str;
}
```

第二个ListBox控件为多选列表。程序中使用foreach循环语句遍历每一个列表项(Items中的每一个ListItem)，如果该项被选中(其Selected属性值为True)，则记录该项的值和显示文本。最后，集中显示所有选中项的信息或“未选”信息。

在浏览器中查看该页面，效果如图4-7所示。

因为将AutoPostBack属性改为了True，所以在两个ListBox控件中做选择后都会引发页面回传。当页面再次加载时，Label控件上会显示对应ListBox的已选内容。



图4-7 ListBox控件的执行效果

与前面介绍的控件相比，ListBox控件有一个重要的

增强，就是可以与数据源绑定。所谓“与数据源绑定”就是可以将从数据库中读出的数据自动加载到控件上来，对于 ListBox 控件来说，就是可以自动添加为 ListBox 的可选项。与数据源绑定的内容将在后面的章节中详细介绍，这里只简单介绍一下 ListBox 控件与数据绑定有关的属性和事件（见表 4-8），供读者使用时查询。

表 4-8　ListBox 控件与数据绑定有关的属性和事件

属性名称	说　明
DataMember	当数据源包含多个不同的数据项列表时，该属性表示数据绑定控件绑定到的数据列表的名称
DataSource	数据源，ListBox 控件从该数据源中检索其数据项列表
DataSourceID	数据源控件 ID，ListBox 控件从该数据源控件中检索其数据项列表
DataTextField	为列表项提供文本内容的数据源字段
DataTextFormatString	格式化字符串，该字符串用来控制如何显示绑定到列表控件的数据
DataValueField	为各列表项提供值的数据源字段
事件名称	**说　明**
DataBinding	当服务器控件绑定到数据源时发生
DataBound	在服务器控件绑定到数据源后发生

4.3.4　DropDownList 控件

DropDownList 控件也是一种列表选择控件，与 ListBox 控件非常相似。但 DropDownList 控件正常情况下只显示一项，单击控件上的按钮时才弹出下拉式列表显示其余的项。另外，DropDownList 控件只能单选。

在 4.3.3 小节 ListBox 控件的示例中，选项都是由程序添加的。其实 ListBox 控件与 DropDownList 控件相同，各选项既可由程序添加，也可以在设计期间添加。

创建一个名为 UseDropDownList 的网站。

从工具箱中拖一个 DropDownList 控件到页面上来，页面上增加如下代码：

```
<asp:DropDownList ID="DropDownList1" runat="server">
</asp:DropDownList>
```

在设计视图上选中 DropDownList 控件，在集成开发环境右下边的属性窗口中单击 Items 属性右侧的省略号按钮，打开“ListItem 集合编辑器”对话框。在其中增加四个选项，分别设置其 Text 和 Value 属性，结果代码如下：

```
<asp:DropDownList ID="DropDownList1" runat="server">
    <asp:ListItem Value="J01">C语言程序设计</asp:ListItem>
    <asp:ListItem Value="J02">数据结构</asp:ListItem>
    <asp:ListItem Value="J03">网络程序设计</asp:ListItem>
    <asp:ListItem Value="J04">.NET程序设计</asp:ListItem>
```

```
</asp:DropDownList>
```

这样，在执行时，DropDownList 中就会有 4 个选项。

4.4 Panel 控件

Panel 控件用于包含其他控件，它提供以下 3 个功能。

(1) 控制所包含控件的可见性。

(2) 控制所包含控件的外观。

(3) 方便以编程方式生成控件。

可以将整个页面划分为几个功能区，每一个功能区为一个包含多个(一组)其他控件的 Panel 控件。这样，便于对各组控件的整体控制(包括隐藏与显示等)，给编程带来了更大的灵活性与简便性。Panel 控件的属性如表 4-9 所示。

表 4-9 Panel 控件所特有的属性

属性名称	说明
BackImageUrl	控件背景图像的 URL
Direction	Panel 控件中所包含控件的排列方向
HorizontalAlign	Panel 控件内容的水平对齐方式
ScrollBars	指定 Panel 控件中滚动条的可见性和位置
Wrap	Panel 控件中的内容是否换行

Direction 属性确定在 Panel 控件中，所包含控件的排列方向，其值是一个 ContentDirection 枚举值，包括以下内容。

- NotSet：未设置内容的方向(默认值)。
- LeftToRight：内容的方向为从左到右。
- RightToLeft：内容的方向为从右到左。

HorizontalAlign 属性表示 Panel 控件内容的水平对齐方式，其值如下。

- Center：容器的内容居中。
- Justify：容器的内容均匀展开，与左右边距对齐。
- Left：容器的内容左对齐。
- NotSet：未设置水平对齐方式(默认值)。
- Right：容器的内容右对齐。

由于 Panel 是一个窗口控件，因此 ScrollBars 属性是它的一个重要属性。该属性指定 Panel 控件中滚动条的可见性和位置，其值是一个 ScrollBars 枚举值，包括：

- None：不显示任何滚动条。
- Horizontal：只显示水平滚动条。
- Vertical：只显示垂直滚动条。

- Both：同时显示水平滚动条和垂直滚动条。
- Auto：自动显示。无必要时(Panel 控件中内容的大小未超出 Panel 控件本身的大小)则不显示任何滚动条;如有必要可根据需要自动显示水平滚动条、垂直滚动条或同时显示这两种滚动条。

创建一个名为 UsePanel 的网站。

在默认的主页上增加一个 Panel 控件、一个 CheckBox 控件、一个 TextBox 控件、一个 Button 控件和文本若干。在 Panel 控件上增加显示文本。相关页面代码如下：

```
<h2>Panel 控件：</h2>
<asp:Panel ID="Panel1" runat="server" Width="80% " BackColor="# E0E0E0"
        BorderStyle="Outset" HorizontalAlign="Center">
    这是一个 Panel 控件。您可以控制它是否显示,或为它动态地增加其他控件。
    <br />
</asp:Panel>
<asp:CheckBox ID="CheckBox1" runat="server" text="隐藏 Panel" />
<br />
包含 Label 数：
<asp:TextBox ID="TextBox1" runat="server" Width="37px">0</asp:TextBox>
<br />
<asp:Button ID="Button1" runat="server" text="刷新页面" />
```

注意：初始时,Panel 控件中并没有其他控件,只有一串说明文字。CheckBox 控件、TextBox 控件和 Button 控件都在 Panel 控件之外,用于对 Panel 进行控制。

将页面的 Page_Load()函数修改为如下代码：

```
protected void Page_Load(object sender,EventArgs e)
{
    if (CheckBox1.Checked)
    {
        //如果 CheckBox1 选中,则不显示 Panel
        Panel1.Visible=False;
    }
    else
    {
        Panel1.Visible=True;
    }

    //取得要生成的 Label 控件数
    int n=Int32.Parse(TextBox1.Text);
    for (int i=1; i<=n; i++)
    {
        //生成新的 Label 控件
        Label lbl=new Label();
        lbl.Text="Label"+(i).ToString();
```

```
            lbl.ID="Label"+(i).ToString();
            //将 Label 加到 Panel 上
            Panel1.Controls.Add(lbl);
            Panel1.Controls.Add(new LiteralControl("<br />"));
        }
    }
```

页面加载时，根据 CheckBox1 是否选中来控制 Panel 控件是否显示；根据 TextBox1 中输入的数值在 Panel 控件上，由程序控制生成相应个数的 Label 控件。没有针对 Button1 进行处理的代码，Button1 的作用是引发网页回送服务器。

页面第一次的执行效果如图 4-8 所示。

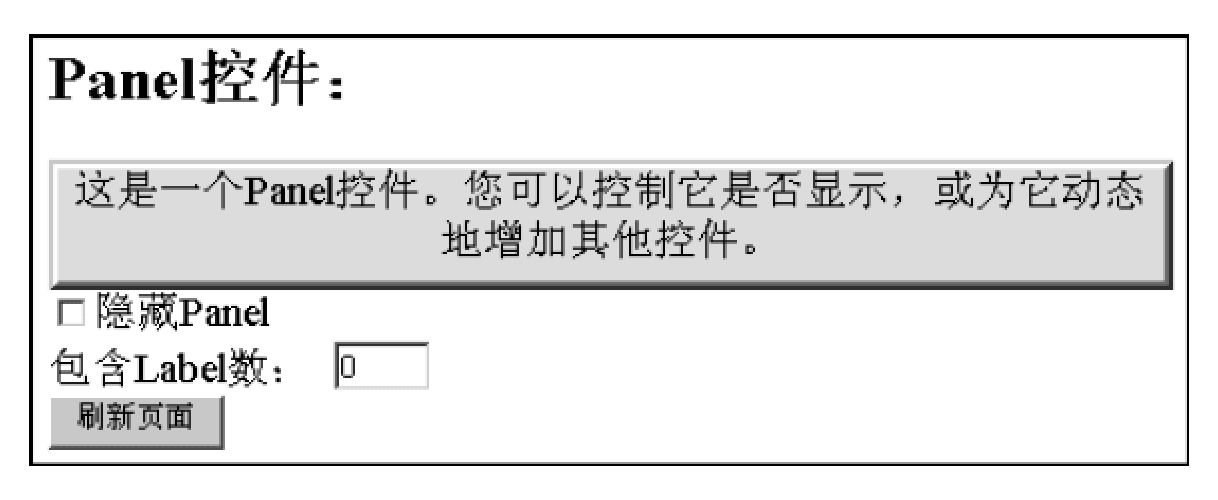


图 4-8　Panel 控件的执行效果(一)

如果选择“隐藏 Panel”复选框，按“刷新页面”按钮，页面重新加载后则不显示 Panel 控件。

如果在“包括 Label 数”输入框中输入一个值，按“刷新页面”按钮，页面重新加载时会在 Panel 控件上自动创建几个 Label 控件，如图 4-9 所示。

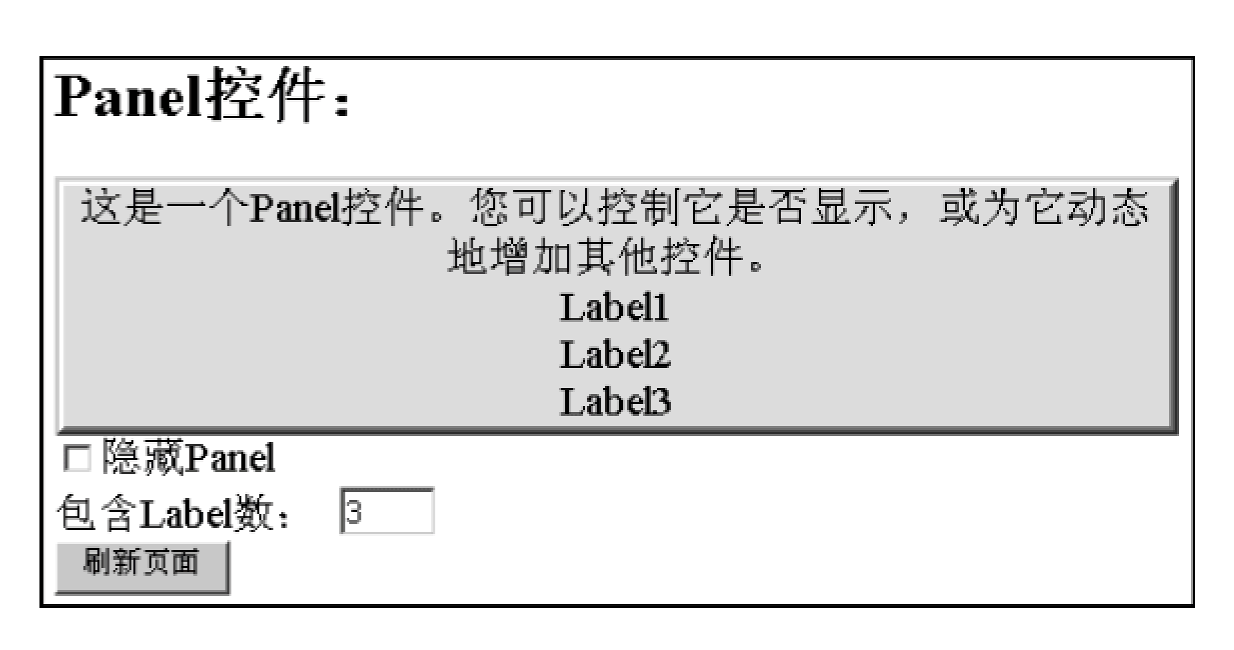


图 4-9　Panel 控件的执行效果(二)

4.5　图片控件

4.5.1　Image 控件

Image 控件用于显示一个图片。Image 控件本身不包含响应用户交互的事件，如果需要将图片作为按钮来使用，可使用前面提到过的 ImageButton 控件。Image 控件所特有的属性如表 4-10 所示。

表 4-10　Image 控件所特有的属性

属性名称	说　明
AlternateText	当图像不可用时，Image 控件中显示的替换文本
ImageAlign	Image 控件相对于网页上其他元素的对齐方式
ImageUrl	要显示图像的 URL

4.5.2　ImageMap 控件

许多网站的主页顶部都是以一个较大的图片作为背景，以动画或静止的方式显示网站的名称等信息。在这个图片之上再有一些小的图片作为导航，单击后可导航到“网站首页”、“会员服务”、“产品介绍”和“与我们联系”等内容。有时为了达到更好的艺术效果，这些小图片的排列还往往是不规则的。

在网页上实现上述效果可以有很多种方法，如果使用 ASP. NET 开发网站应用，则有一个非常方便的选择，那就是使用 ImageMap 控件，其所特有的属性和事件如表 4-11 所示。

表 4-11　ImageMap 控件所特有的属性和事件

属性名称	说　明
AlternateText	当图像不可用时，Image 控件中显示的替换文本
ImageAlign	Image 控件相对于网页上其他元素的对齐方式
ImageUrl	要显示的图像的 URL
HotSpotMode	单击 ImageMap 控件的作用点时，HotSpot 对象的默认行为
HotSpots	HotSpot 对象的集合，这些对象表示 ImageMap 控件中定义的作用点
事件名称	**说　明**
Click	单击 ImageMap 控件的 HotSpot 对象时发生

利用 ImageMap 控件可以显示一个图像，该图像包含多个用户可以单击的区域，这些区域称为作用点。每一个作用点都可以是一个单独的超链接或网页回送事件。使用 ImageMap 控件完成上述网页功能，其工作可简化为：制作一个漂亮的、完整的图片，为图片上合适的区域定义作用点。

ImageMap 控件主要由两部分组成。第一部分是图像，它可以是任何标准 Web 图形格式的图形，如 . gif、. jpg 或 . png 文件等。

第二部分是作用点控件（HotSpot 对象）的集合。每个作用点控件都是一个不同的元素。对于每个作用点控件都需要指定其形状（圆形、矩形或多边形）以及其位置和大小。例如，如果创建一个圆形作用点，则应定义圆心的 x 和 y 坐标以及圆的半径。

可以指定在用户单击 ImageMap 控件上的某个作用点时发生的事件。可以将每个作用点配置为超链接，也可以将作用点配置为在用户单击时回送网页。为每个作用点提

供一个唯一值，在服务器端处理 ImageMap 控件的 Click 事件时，可以读取分配给每个作用点的唯一值。

HotSpots 是一个 HotSpotCollection 对象，它是 HotSpot 对象的集合，每个 HotSpot 对象表示 ImageMap 控件中定义的一个作用点。使用 HotSpots 属性，可以用编程的方式管理 ImageMap 控件中的作用点，包括添加、插入、移除或检索等。与 Items 属性一样，HotSpots 属性也是集合类型，它们的编程方法非常相似，这里不再重复，Items 属性编程的内容请参阅 4.3.3 小节。

虽说 HotSpots 是 HotSpot 对象的集合，但由于 HotSpot 是抽象类，因此无法直接创建 HotSpot 实例。但是，可以使用 CircleHotSpot、RectangleHotSpot 和 PolygonHotSpot 类定义作用点。在这些类中，除了表示形状的属性外，都有一个 HotSpotMode 属性

单击 ImageMap 控件中的 HotSpot 时，页面可以导航至一个 URL、生成一个到服务器的回发或是不执行任何操作。到底如何处理，由 HotSpotMode 属性值确定。HotSpotMode 属性的可能值如下。

- NotSet：HotSpot 使用由 ImageMap 控件的 HotSpotMode 属性设置的行为。
- Inactive：HotSpot 不具有任何行为。
- Navigate：定位到 URL(默认值)。
- PostBack：回送服务器。

可以在 ImageMap 控件的 HotSpotMode 属性上或是在每个单独的 HotSpot 对象的 HotSpotMode 属性上指定 HotSpot 行为。如果同时设置这两个属性，那么每个单独的 HotSpot 对象上指定的 HotSpotMode 属性优先。

创建一个名为 UseImageMap 的网站。

在集成开发环境右上部的“解决方案资源管理器”中的网站名称上单击右键，在弹出式菜单中选择“新建文件夹”，可为当前网站创建一个子文件夹。在文件夹上单击右键，在弹出式菜单中选择“添加现有项”，可将现存的文件引入到网站的当前位置(文件夹)中来。例如，可以用上述方式为当前网站创建一个名为 images 的文件夹，并将一个称为 title1.jpg 的图片文件引入到该文件夹中来。

在页面上增加一个 ImageMap 控件，将其 ImageUrl 属性改为"images\title1.jpg"，将其 HotSpotMode 属性值改为 Navigate。

在设计视图上选中 ImageMap 控件，在集成开发环境右下部的属性窗口中单击 HotSpots 属性右侧的省略号按钮，打开“HotSpot 集合编辑器”对话框，如图 4-10 所示。

HotSpot 集合编辑器左下部的“添加”按钮右边有一个向下的箭头，表示该按钮包含下拉式列表，可选择增加不同类型的作用点。增加三个 RectangleHotSpot 类型的作用点，分别设置其 Bottom、Top、Left 和 Right 属性，其他属性的设置如表 4-12 所示。

在设计视图中双击 ImageMap 控件，系统自动为其创建 Click 事件处理函数。

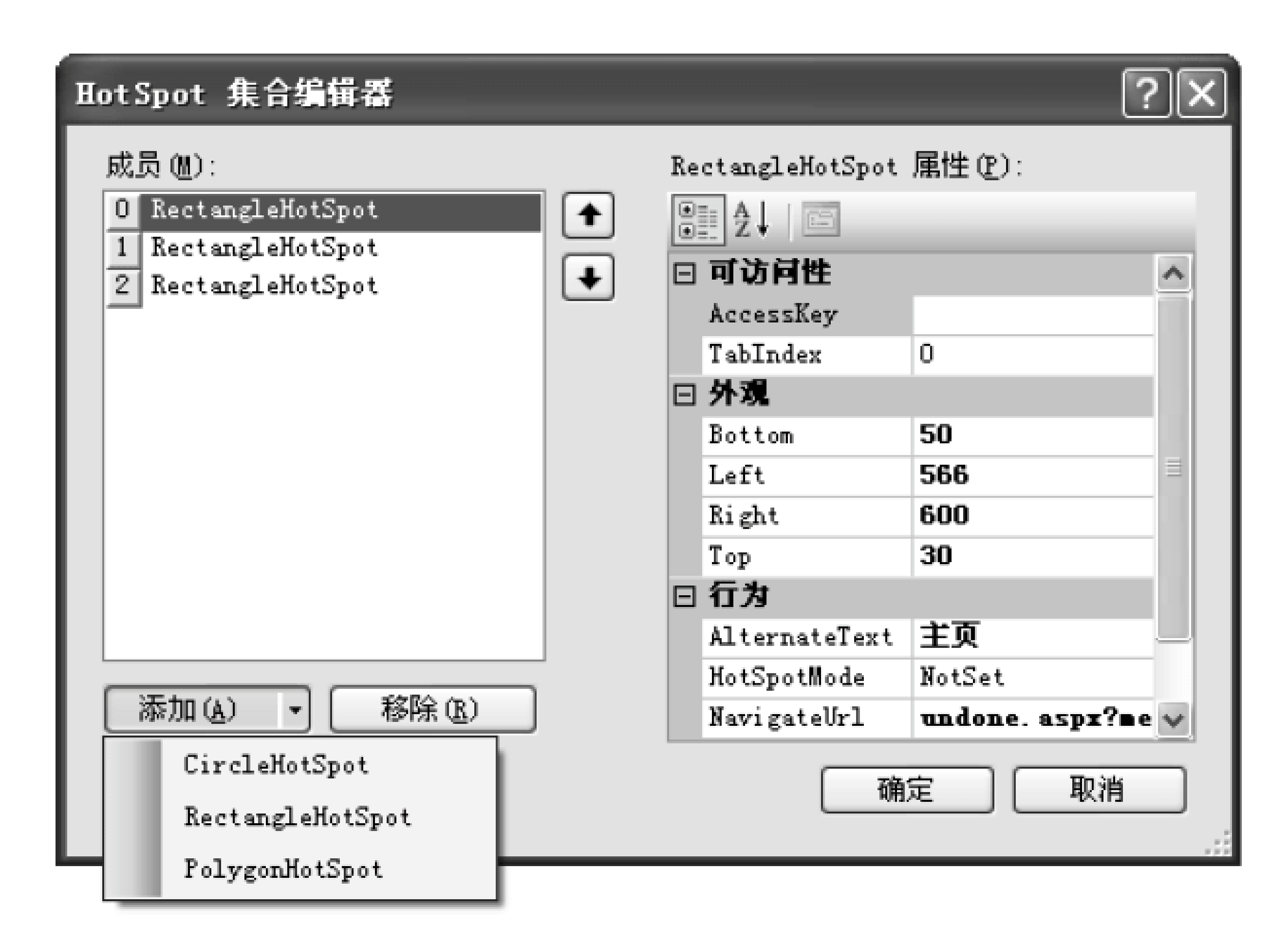


图 4-10 "HotSpot 集合编辑器"对话框

表 4-12 示例中增加的作用点及其属性

RectangleHotSpot1	
AlternateText	主页
NavigateUrl	undone.aspx? mess=主页
RectangleHotSpot2	
HotSpotMode	PostBack
PostBackValue	contactus
AlternateText	联系我们
RectangleHotSpot3	
HotSpotMode	PostBack
PostBackValue	help
AlternateText	帮助

经过上述操作,相关页面代码如下:

```
<asp:ImageMap ID="imgmapTitle" runat="server"
    ImageUrl="images\title1.jpg"
    HotSpotMode="Navigate" OnClick="imgmapTitle_Click">
  <asp:RectangleHotSpot
    Bottom="50" Top="30" Left="566" Right="600"
    AlternateText="主页"
    NavigateUrl="undone.aspx? mess=主页"/>
  <asp:RectangleHotSpot
    HotSpotMode="PostBack"
    PostBackValue="contactus"
```

```
            Bottom="50" Top="30" Left="612" Right="676"
            AlternateText="联系我们" />
        <asp:RectangleHotSpot
            HotSpotMode="PostBack"
            PostBackValue="help"
            Bottom="50" Top="30" Left="690" Right="722"
            AlternateText="帮助" />
</asp:ImageMap>
```

将 Click 事件处理函数修改为如下代码：

```
protected void imgmapTitle_Click(object sender,ImageMapEventArgs e)
{
    if (e.PostBackValue=="contactus")
        Response.Redirect("undone.aspx?mess=联系我们");
    if (e.PostBackValue=="help")
        Response.Redirect("undone.aspx?mess=帮助");
}
```

注意：ImageMap 控件的 Click 事件处理函数的第 2 个参数是 ImageMapEventArgs 类型对象，该对象包含一个 PostBackValue 属性，用于获得引发回送事件的作用点的 PostBackValue 值，程序可根据该值作出相应的处理。有关事件处理函数参数的一般性讨论见 4.1 节。

页面执行效果如图 4-11 所示。



图 4-11　ImageMap 控件的执行效果

示例图片使用的是本书应用实例的标题图片，上面包括三个超链，分别是：主页、联系我们和帮助。执行时当鼠标移动到图片的这三个作用点上面时就会变成小手形状，单击鼠标，系统就会导航到目的页面。

从结果上看，单击上述三个作用点都是导航到 undone.aspx 页面，只是传递了不同的参数。但在处理上还是有差别的。当单击"主页"时，页面直接导航到目的页面。当单击另两个作用点时，页面回传到服务器，在服务器端再重定位到目的页面；当然，在服务器端也可以进行其他任何处理。

提示：要正确运行上述网站，还需要创建一个称为 undone.aspx 的页面作为目的页面。该页面接受一个参数 mess，并将参数值显示在一个 Label 控件上。相关的页面代码如下：

```
<asp:Label ID="Message" runat="server" Font-Names="华文新魏" Font-Size="36pt"
    ForeColor="Blue"></asp:Label>
<br />抱歉，此功能尚未完成。
```

Page_Load()函数的相关代码如下：

```
if (!Page.IsPostBack)
{    ///显示数据
     Message.Text=Request.Params["mess"].ToString();
}
```

有关向页面传递参数的内容将在第 6 章详细介绍。

习　　题

1. 在 ASP.NET 中，Web 控件共有哪几种类型？
2. 与传统的客户端 HTML 控件相比，ASP.NET 控件有哪些方面的改进？
3. 很多控件具有 AutoPostBack 属性，请概述该属性的作用。
4. ASP.NET 事件可能有几个参数？请对这些参数做一个简要说明。
5. 创建一个空白 HTML 文档，使用 Label 控件让页面显示文字内容“这是一个 Label 控件。”。
6. 在题 2 中所实现的页面上增加一个 Button 按钮，要求当按下该按钮后能够改变 Label 控件上的文字字体和颜色。
7. TextBox 控件有几种行为模式？其行为模式由哪个属性确定？
8. 创建一个 HTML 文档，窗体上有一个 TextBox 控件和一个 Button 控件。要求每当用户单击按钮时，文本框会显示数字，反映单击的次数。
9. 创建一个空白 HTML 文档，分别使用 HyperLink 控件和传统的标记建立超链，运行该文档，观察两种实现方法的执行效果。
10. 请简要说明 CheckBox 控件和 RadioButton 控件的区别，并在一个空白页面上显示这两个控件。
11. ListBox 控件有几种选择模式？选择模式由哪个属性决定？
12. 简述 ListBox 控件的 Items 属性的编程方法。
13. 编写一段程序，分别实现向 ListBox 控件 ListBox1 和 DropDownList 控件 DropDownList1 中自动添加 10 个数，每个数占一项。
14. ListBox 控件有哪些与数据绑定有关的属性和方法？
15. 简述 Panel 控件的功能。
16. 创建一个 HTML 文档，实现如图 4-8 所示的执行效果。
17. 什么情况下适合使用 ImageMap 控件？
18. 参照 4.5 节的介绍，创建一个 HTML 文档，实现如图 4-11 所示的页面效果，图片自选。

第 5 章 chapter 5

ASP.NET 高级控件

第 4 章介绍的 ASP.NET 控件基本上都能在传统的 HTML 控件中找到其原型，但是将大部分处理操作都转移到了服务器端进行，这样可以得到更强的控制能力和编程方便性。除第 4 章介绍的控件之外，ASP.NET 还提供了大量功能更完整、更有针对性的高级控件。高级控件一般都跟具体的功能有关，使用一个高级控件就能够完成一项任务的核心功能，这样就可以较大地减少编程的工作量。

本章仅选择性地介绍 Calendar、FileUpload、Wizard、PlaceHolder、AdRotator 和验证控件等。

5.1 Calendar 控件

5.1.1 Calendar 控件的基本概念

以前开发基于浏览器的应用程序时，有关日期型数据的操作(如输入、选择等)，因为涉及格式、初值和校验等多方面的内容，程序员往往需要花费大量的精力对其进行处理。为了解决这个问题，许多程序员开发了封装良好的脚本控件提供给大家，但这些控件外观、接口、功能各异，继承和定制都很困难。

ASP.NET 中的 Calendar 控件很好地解决了这个问题，使与时间有关的编程不再困难。

Calendar 是一个功能丰富的控件，很多与日期有关的功能都可以以该控件为基础创建。Calendar 控件本身的功能主要包括：

- 显示一个日历，包括一个月的详细日历和其他一些相关信息。
- 允许用户选择一天、一周或一个月。
- 允许用户移到下一个月或上一个月。
- 以编程方式控制选定日期的显示。

创建一个名为 Calendar 的网站。

从工具箱中拖一个 Calendar 控件到 Default.asp 页面上，页面上会增加如下代码：

```
<asp: Calendar ID= "Calendar1" runat= "server">< /asp: Calendar>
```

执行该页面,效果如图 5-1 所示。

这个界面已经很专业了,除了一次拖动之外,还没有修改任何代码。

Calendar 控件可以通过属性和事件来进行定制。从表 5-1 可以看出,Calendar 控件提供了丰富的属性和事件处理,使编程人员可以控制 Calendar 控件的几乎所有细节。

<			2007年4月			>
日	一	二	三	四	五	六
25	26	27	28	29	30	31
1	2	3	4	5	6	7
8	9	10	11	12	13	14
15	16	17	18	19	20	21
22	23	24	25	26	27	28
29	30	1	2	3	4	5

图 5-1 Calendar 控件的执行效果(一)

表 5-1 Calendar 控件所特有的属性和事件

属性名称	说明
Caption	日历标题
CaptionAlign	日历标题的对齐方式
CellPadding	单元格的内容和单元格边框之间的间隔,以像素为单位
DayHeaderStyle	显示一周中某天部分的样式属性
DayNameFormat	一周中各天的名称格式
DayStyle	本月日期的样式
FirstDayOfWeek	Calendar 控件的第一列显示一周中的哪天
NextMonthText	“下一月”导航元素的文本
NextPrevFormat	Calendar 控件的标题部分中,“下一月”和“上一月”导航元素的格式
NextPrevStyle	“下一月”和“上一月”导航元素的样式
OtherMonthDayStyle	Calendar 控件上非本月日期的样式
PrevMonthText	“上一月”导航元素的文本
SelectedDate	选定的日期
SelectedDates	System. DateTime 对象的集合,这些对象表示 Calendar 控件上的选定日期
SelectedDayStyle	选定日期的样式
SelectionMode	Calendar 控件的日期选择模式,该模式指定用户可以选择单日、一周还是整月
SelectMonthText	选择器列中,月份选择元素的文本
SelectorStyle	周和月选择器列的样式
SelectWeekText	选择器列中,周选择元素的文本
ShowDayHeader	是否显示一周中各天的标头
ShowGridLines	是否用网格线分隔 Calendar 控件上的日期
ShowNextPrevMonth	是否在标题部分显示“下一月”和“上一月”导航元素
ShowTitle	是否显示标题部分

续表

属性名称	说　明
TitleFormat	标题部分的格式
TitleStyle	标题部分的样式
TodayDayStyle	Calendar 控件上“今天”日期的样式
TodaysDate	今天的日期值
VisibleDate	要在 Calendar 控件上显示的月份
WeekendDayStyle	周末日期的样式
事件名称	**说　明**
DayRender	当为 Calendar 控件在控件层次结构中创建每一天时发生
SelectionChanged	当用户单击日期选择器选择了一天、一周或整月时发生
VisibleMonthChanged	当用户单击标题部分的“下一月”或“上一月”导航元素时发生

5.1.2 改变 Calendar 控件的外观

可以通过简单地改变 Calendar 控件的属性来得到丰富的外观表现形式。

在页面上再增加一个 Calendar 控件，可参照下面示例代码直接修改页面代码，也可以在设计视图中逐项修改属性值。示例代码如下：

```
<asp:Calendar ID="Calendar2" runat="server" BackColor="#COFFFF"
    BorderColor="#00C0C0" BorderWidth="1px" Font-Names="Verdana"
    Font-Size="9pt" ForeColor="Black" Height="100px"
    NextPrevFormat="FullMonth" Width="202px" FirstDayOfWeek="Monday"
    SelectionMode="DayWeekMonth" SelectMonthText="月 &gt;"
    SelectWeekText="周 &gt;" ShowGridLines="True">
    <SelectedDayStyle BackColor="#333399" ForeColor="White" />
    <TodayDayStyle BackColor="#CCCCCC" />
    <OtherMonthDayStyle ForeColor="#999999" />
    <NextPrevStyle Font-Bold="True" Font-Size="8pt"
        ForeColor="#333333" VerticalAlign="Bottom" />
    <DayHeaderStyle Font-Bold="True" Font-Size="8pt" />
    <TitleStyle BackColor="White" BorderColor="Black" BorderWidth="4px"
        Font-Bold="True" Font-Size="12pt" ForeColor="#333399" />
</asp:Calendar>
```

在上面的示例中配置了很多属性，在此仅详细介绍一下 SelectionMode 属性的相关内容。

通过改变 SelectionMode 属性可以控制日历的选择模式，该属性的可选值包括：

- Day(默认值)：允许选择单个日期。
- DayWeek：允许选择单个日期或整周。

- DayWeekMonth：允许选择单个日期、整周或整月。



图 5-2　Calendar 控件的执行效果(二)

- None：不允许选择日期，只能导航。

在本例中，将 SelectionMode 属性设为了 DayWeekMonth，并设置 SelectMonthText ＝" 月 >" SelectWeekText ＝" 周 >"，用户就可以通过日历左部的“月＞”超链选择整月，或通过“周＞”超链选择整周了。

执行效果如图 5-2 所示。

这仅仅是改变外观的一个例子，Calendar 控件在实际应用中还有很多发挥空间。

5.1.3　对 Calendar 控件编程

Calendar 控件提供了三个特有的可编程事件，分别在不同的时机触发。通过对它们进行编程，可实现具有极强用户交互性的日期相关功能。本节仅介绍 SelectionChanged 事件的编程。

当用户在 Calendar 控件中选择一天、整周或整月时，将触发 SelectionChanged 事件。

在 5.1.2 小节的页面上再增加三个 Label 控件。在设计视图中双击日历控件，系统会自动打开源代码文件并将光标停留在函数 Calendar2_SelectionChanged()内。

此时页面代码的相关部分为：

```
<asp: Calendar…
    OnSelectionChanged="Calendar2_SelectionChanged">
    …
</asp: Calendar>
<asp: Label ID="Label1" runat="server"></asp: Label><br/>
<asp: Label ID="Label2" runat="server"></asp: Label><br/>
<asp: Label ID="Label3" runat="server"></asp: Label><br/>
```

在函数 Calendar2_SelectionChanged()内输入代码：

```
Label1.Text="今天的日期是："+
    Calendar2.TodaysDate.ToShortDateString();
if (Calendar2.SelectedDate != DateTime.MinValue)
    Label2.Text="选择的开始日期是："+
        Calendar2.SelectedDate.ToShortDateString();
Label3.Text="选择的天数是："+
    Calendar2.SelectedDates.Count.ToString();
```

三月 2007年4月 五月

今天的日期是：2007-4-28
选择的开始日期是：2007-4-16
选择的天数是：7

图 5-3　Calendar 控件的执行效果(三)

页面执行时显示当月的日历，当从左侧的选择器列中单击“周＞”超链选择一周时，页面重新加载，效果如图 5-3 所示。

在本书应用实例中，使用 Calendar 控件实现日程安排功能，还在需要输入日期数据时，用 Calendar 控件编程实现弹

出对话框供用户选择日期。

5.2 FileUpload控件

应用程序中经常需要将文件上传到服务器。如本书的应用实例中，教师就需要将课件上传到服务器供学生下载学习。

以前可以使用HTML控件＜input id＝"File1" type＝"file"/＞来完成文件上传功能。ASP.NET提供了FileUpload控件，可以达到更好的功能效果，FileUpload控件所特有的属性和方法见表5-2。

表5-2 FileUpload控件所特有的属性和方法

属性名称	说明
FileBytes	从使用FileUpload控件上传的文件返回一个字节数组
FileContent	Stream对象，它指向上传的文件
FileName	上传文件的名称(不包含此文件在客户端的文件路径)
HasFile	FileUpload控件是否包含文件
PostedFile	上传文件的基础HttpPostedFile对象
方法名称	**说明**
Focus	为控件设置输入焦点
SaveAs	将上传文件的内容保存到Web服务器上指定的目录中

FileUpload控件在客户端表现为一个文本输入框和一个浏览按钮，供用户选择本地文件。包含单一FileUpload控件的页面显示效果如图5-4所示。

图5-4 FileUpload控件的执行效果

用户选择了要上传的文件后，FileUpload控件不会自动将该文件上传到服务器。程序员必须显式地控制文件的提交，例如，可以提供一个“上传”按钮，用户单击该按钮时即可提交上传文件。文件上传到服务器后也不会自动保存，仍然需要程序员编程进行处理。如果提供了一个用于提交文件的按钮，在服务器端处理上传文件的代码就可以放在该按钮的单击事件处理函数中。

FileUpload控件提供了丰富灵活的文件处理功能。

首先可以通过HasFile属性判断是否有上传文件。如果有就可以通过FileName属性获得上传文件的名称。

处理文件可以有多种方法。可以调用FileUpload控件的SaveAs方法将上传文件的内容保存到Web服务器上指定的目录中；也可以通过FileBytes属性获得文件的二进制内容，并将内容保存到字节数组中；也可以通过FileContent属性将上传的文件当做流来处理。后两种方法可对文件内容进行直接操作，虽然操作复杂，但可以得到更丰富的功

能，如将文件内容直接存储到数据库中。

FileUpload 控件还提供了一个 PostedFile 属性，它的类型是 HttpPostedFile 对象，通过它也可以对上传的文件进行操作，其成员和方法如表 5-3 所示。

表 5-3 HttpPostedFile 对象的成员和方法

成员名称	说明
ContentLength	上传文件的大小(以字节为单位)
ContentType	上传文件的 MIME 内容类型
FileName	上传文件在客户端的完全限定名称(包含此文件在客户端的文件路径)
InputStream	Stream 对象，它指向上传的文件(与 FileUpload 控件的 FileContent 属性相同)
方法名称	**说明**
SaveAs	将上传文件的内容保存到 Web 服务器上的指定目录中(与 FileUpload 控件的 SaveAs 方法作用相同)

创建一个名为 FileUpload 的网站。为网站创建一个新的文件夹，如 Uploads。

向页面上拖放一个 FileUpload 控件、一个 Button 控件和一个 Label 控件，页面代码如下：

```
<asp:FileUpload ID="FileUpload1" runat="server"/>
<br />
<asp:Button ID="Button1" runat="server" Text="上传"
    OnClick="Button1_Click" /><br/>
<asp:Label ID="Label1" runat="server"></asp:Label>
```

修改按钮的单击事件处理函数，代码如下：

```
protected void Button1_Click(object sender,EventArgs e)
{
    string str="";
    //如果 FileUpload控件包含文件
    if (FileUpload1.HasFile)
    {
        try
        {
            //生成完整的文件名：绝对路径+文件名
            string fn=Server.MapPath(Request.ApplicationPath)+
                "\\Uploads\\"+FileUpload1.FileName;
            //保存文件
            FileUpload1.SaveAs(fn);

            //如果保存成功,生成结果信息
            str+="客户端文件："+FileUpload1.PostedFile.FileName;
            str+="<br/>服务器端保存："+fn;
```

```
            str+="<br/>文件类型："+FileUpload1.PostedFile.ContentType;
            str+="<br/>文件大小："+
                FileUpload1.PostedFile.ContentLength;
        }
        catch (Exception ex)
        {
            //如果文件保存时发生异常,则显示异常信息
            str+="保存文件出错："+ex.Message;
        }
    }
    else
    {
        //如果不包含文件,给出提示
        str="无上传文件。";
    }
    Label1.Text=str;
}
```

程序中先根据 FileUpload1 控件的 HasFile 属性判断是否有上传文件。如果有上传文件,则先生成服务器端保存文件的完整文件名(有关 Server 对象的内容请参阅第 6 章的 6.6 节),再调用控件的 SaveAs 方法保存文件。然后,通过 PostedFile 属性来获得文件信息。

执行界面如图 5-5 所示。

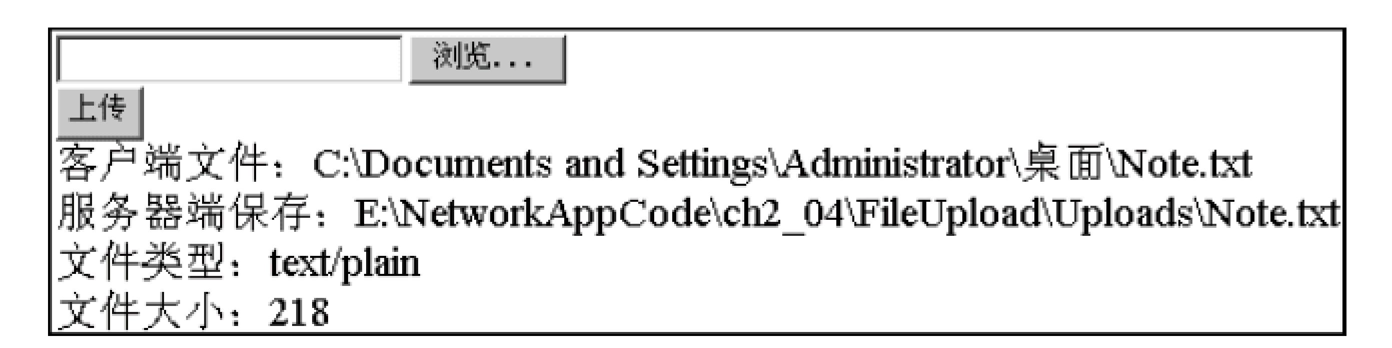


图 5-5　FileUpload 控件的执行效果

页面第一次执行时没有下面的信息。单击“浏览”按钮选择一个本地文件,再单击“上传”按钮,页面重新加载。如果文件上传成功,会显示如图 5-5 所示的信息。还可以查看网站的 Uploads 子目录,在其中应该能够看到上传的文件。

5.3　Wizard 控件

应用程序经常需要为用户提供向导功能(尽管本书的应用实例没有用到),以引导用户完成多步操作,如收集用户输入等。以前要实现类似的功能非常繁琐,主要因为不同的页面保持相同的风格,及实现在各页面间自由转换并记录状态非常麻烦。

ASP.NET 提供了 Wizard 控件,为用户提供了完成多个步骤操作的实现方法,并方便地在各步骤之间前后导航。

Wizard 控件提供了一种简单的机制,允许轻松地生成步骤、添加新步骤或重新安排

步骤。无需编写代码即可生成线性(从一步转到下一步或上一步)和非线性(从一步转到任意其他步)的导航。该控件能够自动创建合适的按钮,例如“下一步”、“上一步”和“完成”等,并允许用户自定义控件的用户导航。通过配置可以使某些步骤只能被导航一次。在默认情况下,Wizard 控件显示一个包含导航链接的工具栏,让用户可以从当前步骤自由转到其他步。

Wizard 控件有非常灵活的属性设置和事件处理,本节仅给出一个示例,涉及 Wizard 控件使用中的一些重要话题,如果需要实现真正的导航功能,读者还需要参阅 MSDN 的相关内容。

创建一个新网站,可命名为 Wizard。

拖一个 Wizard 控件到页面上,系统会自动增加如下代码:

```
<asp:Wizard ID="Wizard1" runat="server">
    <WizardSteps>
        <asp:WizardStep ID="WizardStep1" runat="server" Title="Step 1">
        </asp:WizardStep>
        <asp:WizardStep ID="WizardStep2" runat="server" Title="Step 2">
        </asp:WizardStep>
    </WizardSteps>
</asp:Wizard>
```

在上述代码中,使用<asp:Wizard>标记声明 Wizard 控件,每个 Wizard 控件又可以嵌套地包含多个导航步,用<asp:WizardStep>标记声明。执行上述页面,可以看到一个有两个步骤的导航程序。这是 Wizard 控件的默认效果,是很不够的。但这也是一个很好的基础,Wizard 控件为用户提供了极其强大的扩展能力。

Wizard 控件内的每个步骤均会给定一个 StepType,用以指示这一步骤是开始步骤、中间步骤还是完成步骤。向导可以根据需要带有任意数量的中间步骤。每个步骤上都可以添加不同的控件(如 TextBox 或 ListBox 控件等)来支持用户操作。当到达 Complete 步骤时,前面输入的所有数据都可以访问。

虽然与其他控件一样,可以在设计视图中对 Wizard 控件进行可视化的设置,但对于 Wizard 控件来说,有些操作还是使用代码的复制比较容易,用户可根据经验,两种方法结合使用。

作为示例,用户可按下述步骤操作。

(1) 改变 Wizard 控件到合适的大小。

(2) 将步骤复制为 5 个,为每个步骤设置 ID 和 Title 属性。

(3) 为每个步骤设置要操作的内容,本示例简化为每步只有一个标题和一段说明文字。

还可以增加一些特殊的导航要求,如:

将第二步的 AllowReturn 属性设为 False,这样在到达第三步时就会只显示“下一步”按钮,也就不能从第三步再回到第二步了。

将第四步的 StepType 属性设为 Finish,这样在到达该步时,就会显示“上一步”和

"完成"按钮。

将第五步的 StepType 属性设为 Complete,当导航到这一步时,左侧的"导航链接工具栏"和导航按钮都不再显示,但前面各步所产生的数据在此都可访问。用户可以在此步骤最终完成工作,并显示一些表示工作完成的信息。

经过上述操作,页面代码如下:

```
<asp:Wizard ID="Wizard1" runat="server" Height="144px" Width="347px">
    <WizardSteps>
        <asp:WizardStep ID="WizardStep1" runat="server" Title="步骤 1">
            <h2>第一步</h2>
            将第一步所要做的工作置于此。
        </asp:WizardStep>
        <asp:WizardStep ID="WizardStep2" runat="server" Title="步骤 2"
            AllowReturn="False">
            <h2>第二步</h2>
            将第二步所要做的工作置于此。
        </asp:WizardStep>
        <asp:WizardStep ID="WizardStep3" runat="server" Title="步骤 3">
            <h2>第三步</h2>
            从这一步不能回退到上一步。
        </asp:WizardStep>
        <asp:WizardStep ID="WizardStep4" runat="server" Title="步骤 4"
            StepType="Finish">
            <h2>第四步</h2>
            将收尾工作置于此。
        </asp:WizardStep>
        <asp:WizardStep ID="WizardStep5" runat="server" Title="完成"
            StepType="Complete">
            <h2>完成</h2>
            工作完成,显示结果信息。
        </asp:WizardStep>
    </WizardSteps>
</asp:Wizard>
```

执行上述页面,可以看出,网站已经具有很专业的导航功能了,但外观还不理想。

用户可以通过手工修改 StepStyle、SideBarStyle 等多个属性来控制外观,也可以简单地套用系统提供的模板:进入设计视图,在智能化标签中选"自动套用格式",选择一种格式,如"简明型"。

再执行页面,可以看到外观已经大不相同。例如,其中第二步的执行效果如图 5-6 所示。

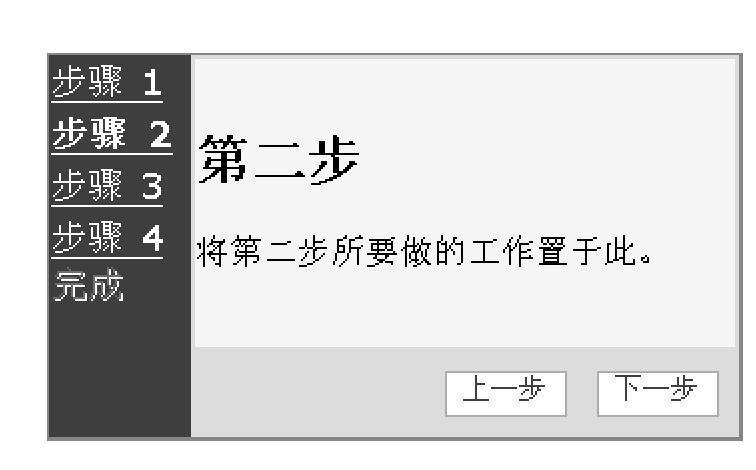


图 5-6 Wizard 控件的执行效果

5.4 PlaceHolder 控件

直到今天,很多程序员仍然习惯于设计静态布局的页面,这能满足大多数的应用要求。但在有些情况下,程序员必须动态地控制页面上都包含哪些控件及这些控件的布局,例如,根据程序运行的上下文关系和用户的权限显示不同的按钮,与当前用户无关的按钮就不再显示,以避免引起混乱。

要达到上述动态效果可以有很多种方法,如使用本书前面介绍的 Panel 控件(参见第 4 章的 4.4 节)。在此,再介绍另一个控件——PlaceHolder 控件,也能达到相似的动态效果。

PlaceHolder 控件也是一个容器控件,它可以被放置在页面上,然后在运行时动态地将子元素(子控件)添加到该容器中,已添加的子元素也可以动态地删除。所不同的是,PlaceHolder 控件是一个空容器,它只呈现其子元素,而没有自己的基于 HTML 的输出。

PlaceHolder 控件的使用方法是:在设计时向页面中增加 PlaceHolder 控件,在运行时向 PlaceHolder 控件添加子控件。

向 PlaceHolder 控件添加子控件的方法是:创建要添加到 PlaceHolder 控件中的某个子控件的实例,调用 PlaceHolder 控件的 Controls 集合的 Add 方法,将子控件加入到 PlaceHolder 中。

创建一个名为 PlaceHolder 的网站。

从工具箱中拖一个 PlaceHolder 控件到页面上,页面上会增加如下代码:

```
<asp:PlaceHolder ID="PlaceHolder1" runat="server"></asp:PlaceHolder>
```

现在执行程序,会显示一个空白页面,上面没有任何内容。

打开当前页面的代码页,在 Page_Load()函数中增加如下代码:

```
Button Button1=new Button();
Button1.ID="HomePage";
Button1.Text="主页";
PlaceHolder1.Controls.Add(Button1);

Button1=new Button();
Button1.ID="ContactUs";
Button1.Text="联系我们";
PlaceHolder1.Controls.Add(Button1);

Button1=new Button();
Button1.ID="Help";
Button1.Text="帮助";
PlaceHolder1.Controls.Add(Button1);
```

再次执行程序，可以看到页面上包含了三个按钮，这三个按钮都是由上述代码在页面加载前在服务器端动态创建的，效果如图5-7所示。

主页 联系我们 帮助

图5-7 在PlaceHolder控件中动态创建的子控件

Panel控件和PlaceHolder控件都是容器控件，都支持动态控件生成，它们有很多相似之处，下面来讨论一下二者的区别。

Panel控件和PlaceHolder控件最根本的区别在于：Panel控件有客户端脚本，而PlaceHolder控件没有，仅在服务器端起分组的作用，这一点可以在客户端浏览器上使用“查看源文件”功能来测试。由此而带来的功能上的差别是，Panel控件具有分组功能和外观功能。

(1) 分组功能：Panel控件可以是静态文本和其他控件的父级控件。通过将一组控件放入一个Panel控件，然后操作该Panel控件，可以将这组控件作为一个整体进行管理。例如，可以通过设置Panel控件的Visible属性来同时隐藏或显示该组所有控件。

(2) 外观功能：Panel控件支持BackColor和BorderWidth等外观属性，可以设置这些属性来为页面上的局部区域创建独特的外观。

由此可见，如果需要在客户端对控件的分组进行操作，则应该使用Panel控件；如果仅在服务器端对分组进行操作，则应该使用PlaceHolder控件。

5.5 AdRotator控件

在商业网站上的广告是必不可少的。其实不只是商业网站，几乎所有的网站都有这样的需求，如宣传自己的产品或服务，友情发布相关网站的一些信息等，这在实现技术上与商业网站的广告是相同的。

ASP.NET提供了AdRotator控件，使用它可以方便地在网页上发布类似广告的信息。网页上的AdRotator控件显示图形图像，当用户单击AdRotator控件时，系统会重定向到指定的目标URL，完成广告导航功能。

AdRotator控件上的广告信息是可变的，在网页每次加载时由系统在广告数据源中随机选择。可以为每条广告加权，以控制该广告被选中的概率，也可以编写在广告间循环的自定义逻辑。

广告数据源可以是XML文件，也可以是数据库表，本节仅介绍XML数据源的使用。数据源中包含广告列表，每一项代表一条广告信息，包括图形文件名和目标URL等。

使用XML数据源的方法是：将每条广告的图像位置、重定向URL及其他一些关联属性写入一个XML文件中，然后通过AdRotator控件的AdvertisementFile属性将AdRotator控件与该文件绑定。在XML文件中，每条广告可以包括下列属性。

- ImageUrl：显示图像的URL。
- NavigateUrl：单击AdRotator控件时要转到的目标URL。
- AlternateText：图像不可用时显示的文本。如果图像可用，当鼠标悬停在图像上时，也会显示该文本。

- Keyword：用于广告筛选的类别。
- Impressions：广告的显示频率值。其值越大，页面加载时被选中的可能性越大，其取值范围为 1～2 048 000 000。
- Height：广告的高度（以像素为单位）。
- Width：广告的宽度（以像素为单位）。

创建一个名为 AdRotator 的网站。

为网站创建一个新的名为 images 的文件夹。将 3 个图片放置到该文件夹中，如 MountainHeart.jpg、ilc.gif 和 jsbike.gif。

提示：可以使用其他任何合适的图片，但需要修改下述 XML 文件中图片文件的名称。

在网站的 App_Data 文件夹中创建一个新的 XML 文件。该文件的扩展名不一定非得是 .xml，为了更加安全、直观，将创建的 XML 文件命名为 BikeAD.ads。

提示：广告文件最好放置在 App_Data 文件夹中，因为 ASP.NET 可防止浏览器利用该文件夹中的文件。

新创建的 XML 文件只有一行代码，将其内容修改为：

```
<?xml version="1.0" encoding="utf-8"?>
<Advertisements>
  <Ad>
    <ImageUrl>~/images/MountainHeart.jpg</ImageUrl>
    <NavigateUrl>http://user.xici.net/b30523/board.asp</NavigateUrl>
    <AlternateText>山地情怀</AlternateText>
    <Impressions>100</Impressions>
  </Ad>
  <Ad>
    <ImageUrl>~/images/ilc.gif</ImageUrl>
    <NavigateUrl>http://www.ilc.net.cn</NavigateUrl>
    <AlternateText>我爱单车</AlternateText>
    <Impressions>50</Impressions>
  </Ad>
  <Ad>
    <ImageUrl>~/images/jsbike.gif</ImageUrl>
    <NavigateUrl>http://bbs.jsbike.com.cn</NavigateUrl>
    <AlternateText>江苏单车网</AlternateText>
    <Impressions>50</Impressions>
  </Ad>
</Advertisements>
```

所有的广告都包含在<Advertisements>元素中，每条广告为一个<Ad>元素。文件中为每一条广告指定了<ImageUrl>、<NavigateUrl>、<AlternateText>和<Impressions>属性。

从工具箱中拖一个 AdRotator 控件到页面上，将其 AdvertisementFile 属性值指定为

刚创建的 XML 文件，将其 Target 属性指定为"_blank"，页面上会增加如下代码：

```
<asp:AdRotator ID="AdRotator1" runat="server"
    AdvertisementFile="App_Data/BikeAD.ads" Target="_blank" />
```

执行该页面，会随机地显示上述 3 个图片中的 1 个。因为没有指定 Height、Width 等属性，因此按原始图片大小显示。单击该图片，系统会导航到指定的网站。多刷新几次页面就会发现，由于“山地情怀”广告的 Impressions 值比其他广告要高，因此被显示的次数也就更多。

如果要自定义广告间循环的逻辑，可以为 AdRotator 控件的 AdCreated 事件创建处理函数，并在该函数中由程序控制选择广告。

由于 AdRotator 控件本身不提供收集统计信息的功能，因此，如果网站要统计各广告的单击次数等信息，则需要编程进行处理，方法是：将所有广告的目的 URL 都指向同一个跟踪页，在该页的 Page_Load()函数中收集需要的统计信息，然后再跳转到广告的目标页上。

5.6 验证控件

对于一个实用的网站，与用户交互是必不可少的。任何网站都不能指望所有用户每一次都能输入正确。一个健全的网站，不但能够避免用户的误操作对系统所造成严重损害，还能够在用户出现操作错误时给出必要的提示，从而帮助用户正确完成操作。因此，对用户的输入进行验证是必要的。

以前完成上述工作往往需要在客户端和服务器端都编写大量的验证代码，不但要花费大量时间和精力，还很难考虑周全，容易出错。ASP. NET 引入的验证控件将验证工作进行了封装，简化了编程人员的工作，提高了程序的可靠性和编程效率。

验证控件可以在客户端直接拦截错误，相应减少了与服务器的交互次数。

ASP. NET 提供了 6 种验证控件，其中 5 个验证控件是由 BaseValidator 类所派生的，它们直接对某个输入控件进行验证；另一个验证控件是 ValidationSummary，它不直接关联输入控件，仅提供了一个集中显示验证错误信息的地方，用于总结来自网页上所有验证控件的错误信息。

由 BaseValidator 类所派生的验证控件如下。

- RequiredFieldValidator：保证用户必须输入某些字段的值。
- CompareValidator：将用户输入到当前控件的值与输入到其他控件的值或常数值进行比较。
- RangeValidator：验证输入值是否在指定范围内。
- RegularExpressionValidator：使用正则表达式来验证输入值。
- CustomValidator：使用自定义的验证程序来验证用户输入。

这些验证控件从 BaseValidator 类继承了一些公共属性，其中最常用的如表 5-4 中所示。

表 5-4　BaseValidator 类的常用属性

属性名称	说　　明
ControlToValidate	要验证的输入控件
Display	验证控件中错误信息的显示行为
EnableClientScript	是否启用客户端验证，默认为 True
ErrorMessage	验证失败时显示时错误信息的文本
ForeColor	验证失败时显示错误信息的颜色，默认为红色
Text	验证失败时显示错误信息的文本

验证控件与被验证控件的关联是通过 ControlToValidate 属性来完成的，由此可见，上述每个验证控件只能对一个输入控件进行验证，而一个输入控件则可以有多个验证控件来验证其结果。

ErrorMessage 属性和 Text 属性都是验证失败时显示错误信息的文本。对于当前验证控件来说，Text 属性的优先级更高，如果同时设置了这两个属性，则仅显示 Text 属性所设置的文本。但如果在页面上填加了 ValidationSummary 控件，在 ValidationSummary 控件中则只收集各验证控件中的 ErrorMessage 属性进行显示。

Display 属性设置验证控件中错误信息的显示行为。其值是 ValidatorDisplay 值之一，包括如下值。

- None：如果想只在 ValidationSummary 控件中显示错误信息，则将 Display 属性值设为 None，错误信息就不会显示在当前验证控件中。
- Static(默认值)：表示不希望网页的布局在验证程序控件显示错误信息时改变，错误信息的显示空间在显示页面时预分配。
- Dynamic：验证失败时在网页上动态放置错误信息，这样可以在未验证之前和验证通过之后节省页面空间。

为了练习使用验证控件，创建一个名为 Validator 的网站。

将网站的主页设计为一个典型的"新用户注册"页面。方法是在自动生成的代码的<div>标记之间增加一个 table，相关代码如下：

```
<div style="text-align: center">
<table width="70% " border="1" style="background-color: silver;">
    <tr>
        <td colspan="2">
            新用户注册<br />
            <asp: Label ID="lbMess" runat="server" Text="请逐项填写注册内容。"
            ForeColor="Blue"></asp: Label>
        </td>
    </tr>
    <tr>
        <td style="width: 40% ">用户名：</td>
```

```
        <td style="width: 60% ; text-align: left;">
            <asp: TextBox ID="UserName" runat="server"></asp: TextBox>
        </td>
    </tr>
    <tr>
        <td>密码：</td>
        <td style="text-align: left;">
            <asp: TextBox ID="Password1" runat="server" TextMode="Password">
            </asp: TextBox>
        </td>
    </tr>
    <tr>
        <td>确认密码：</td>
        <td style="text-align: left;">
            <asp: TextBox ID="Password2" runat="server" TextMode="Password">
            </asp: TextBox>
        </td>
    </tr>
    <tr>
        <td>身份：</td>
        <td style="text-align: left;">
            <asp: DropDownList ID="UserType" runat="server">
                <asp: ListItem>--请选择身份--</asp: ListItem>
                <asp: ListItem Value="学生">学生</asp: ListItem>
                <asp: ListItem Value="教师">教师</asp: ListItem>
                <asp: ListItem Value="管理人员">管理人员</asp: ListItem>
            </asp: DropDownList>
        </td>
    </tr>
    <tr>
        <td>性别：</td>
        <td style="text-align: left;">
            <asp: RadioButtonList ID="Sex" runat="server"
                    RepeatDirection="Horizontal">
                <asp: ListItem>男士</asp: ListItem>
                <asp: ListItem>女士</asp: ListItem>
            </asp: RadioButtonList>
        </td>
    </tr>
    <tr>
        <td>年龄：</td>
        <td style="text-align: left;">
            <asp: TextBox ID="Age" runat="server"></asp: TextBox>
        </td>
```

```
        </tr>
        <tr>
            <td>Email: </td>
            <td style="text-align: left;">
                <ASP: TextBox id="Email" runat="server"  width="228px" />
            </td>
        </tr>
        <tr>
            <td colspan="2">
                <asp: Button ID="Submit" runat="server"
    Text="提交" />
            </td>
        </tr>
    </table>
    </div>
```

执行该页面,效果如图 5-8 所示。

新用户注册页面是一个典型的需要对用户输入进行验证的页面。该页面目前还没有验证功能,将在下面的各小节中为其增加各种验证功能。

图 5-8　新用户注册页面示例

5.6.1　RequiredFieldValidator

RequiredFieldValidator 控件保证输入项不能为空,其使用比较简单。

下面为前述页面的用户名、身份和性别字段(代表了不同的输入控件类型)增加“不可为空”验证。

为网站新增一个页面,名为 RequiredFieldValidator.aspx,将 Default.aspx 页面中由<div>标记所包含的代码复制过来。

在用户名输入框的后面增加一个 RequiredFieldValidator 控件,修改其 ID、ErrorMessage 和 ControlToValidate 属性,代码如下:

```
<asp: RequiredFieldValidator ID="reqUserName" runat="server"
    ErrorMessage="用户名不能为空。" ControlToValidate="UserName">
</asp: RequiredFieldValidator>
```

为身份和性别分别增加 RequiredFieldValidator 验证控件,代码如下:

```
<asp: RequiredFieldValidator ID="reqUserType" runat="server" ErrorMessage="请选择用户身份。"
ControlToValidate="UserType" InitialValue="--请选择身份--">
</asp: RequiredFieldValidator>
……
<asp: RequiredFieldValidator ID="reqSex" runat="server" ErrorMessage="请选择性别。"
ControlToValidate="Sex" Display="Dynamic">
</asp: RequiredFieldValidator>
```

验证控件 reqUserType 对(从下拉式列表控件)输入的用户身份进行验证,所以还要设置 InitialValue 属性。InitialValue 属性是 RequiredFieldValidator 控件的特有属性,表示输入控件的初始值,只有输入控件在失去焦点时的值与此初始值匹配时,验证才失败。

执行该页面,效果如图 5-9 所示。

图 5-9　使用 RequiredFieldValidator 控件的新用户注册页面

5.6.2　ValidationSummary 控件及验证结果判断

5.6.1 节介绍了 RequiredFieldValidator 控件。在介绍其他由 BaseValidator 类所派生的验证控件之前,先介绍一下 ValidationSummary 控件。

在 5.6.1 小节的页面上,同时对多个控件进行验证。实际的应用中,在一个信息收集页面上可能需要对更多的输入控件进行验证。如果采用 5.6.1 小节的方法(每个验证控件单独显示出错信息),用户可能会觉着信息比较零乱,页面的布局设计也比较困难。这时就可以考虑使用 ValidationSummary 控件。ValidationSummary 控件不直接对具体的输入控件进行验证,仅提供一个集中显示验证错误信息的地方,其常用属性如表 5-5 所示。

表 5-5　ValidationSummary 控件的常用属性

属性名称	说　明
DisplayMode	验证摘要的显示模式
HeaderText	ValidationSummary 控件的标题文本
ShowMessageBox	是否弹出一个消息框来显示验证摘要
ShowSummary	是否在网页上显示验证摘要

DisplayMode 属性确定验证信息的显示格式,可以是以下枚举值之一。

- BulletList(默认值):将各项列表显示,带项目符号。
- List:将各项列表显示,不带项目符号。
- SingleParagraph:不列表,将各项连续地显示在同一个段落中。

为网站新增一个名为 ValidationSummary.aspx 的页面,将 RequiredFieldValidator.aspx 页面中由＜div＞标记所包含的代码复制过来。

在"提交"按钮的后面为 table 增加一个新行,其中包含一个 ValidationSummary 控

件，新增的代码如下：

```
<tr>
    <td colspan="2">
        <asp:ValidationSummary ID="ValidationSummary1" runat="server"
            HeaderText="输入验证未通过：" />
    </td>
</tr>
```

可以看出，ValidationSummary 控件没有与任何输入控件相关联，也没有与任何验证控件相关联，只设置了 HeaderText 属性值。

执行该页面，发现 ValidationSummary 控件上已经能够显示所有的验证出错信息。这说明：只要在页面上放置一个 ValidationSummary 控件，不需要做任何关联，它就能自动收集整个页面上所有验证控件的验证出错信息。

但同时，原有的验证控件仍然在显示独立的出错信息，这与简化页面布局的初衷不符。将其他所有验证控件的 Display 属性值都改为 None，重新执行页面，可以看到其他验证控件不再独立显示出错信息，界面如图 5-10 所示。

图 5-10　使用 ValidationSummary 控件的新用户注册页面

除了由验证控件显示输入出错信息外，还可以在程序中对验证结果进行判断。为“提交”按钮增加单击事件处理函数，其程序代码如下：

```
if (Page.IsValid)
{
    lbMess.Text="验证通过。";
}
else
{
    lbMess.Text="注册信息验证未通过。";
}
```

提示：一般情况下，每个页面在服务器端执行前都会被编译为一个从 Page 类所派生的类，相关介绍见 6.1.5 小节。

上述代码对 Page 类的 IsValid 属性进行判断，以确定所有验证是否都获得通过。

如果用户使用的浏览器支持 DHTML，如 Microsoft Internet Explorer 4.0 及更高版本，重新执行该页面。可以看到，如果所有验证都通过，单击“提交”按钮后，页面重新加载，原来显示“请逐项填写注册内容。”的地方显示的是“验证通过。”。但如果有验证没有通过，页面并没有重新加载，也没有像程序中所期望的那样显示“注册信息验证未通过：”。这说明页面并没有回送服务器，而是直接在客户端完成了验证。

如果用户使用的浏览器不支持 DHTML，则每次验证都要回送到服务器端进行，则

会达到程序代码中所期望的效果。

提示：即使是支持 DHTML 的浏览器，有些验证控件的验证处理也可能在服务器端进行，如使用下面将要介绍的 CustomValidator 验证控件。

对于任何输入验证控件，都可以通过将其 EnableClientScript 属性值设为 False（默认为 True）来强制在服务器端进行验证。

下面继续介绍其他几种输入验证控件，如果仅想对输入验证有一个概要了解，前面的内容已经足够了，可以跳过下面几个小节。

RequiredFieldValidator 控件对输入控件有无输入进行判断，下面的几个控件可对输入值进行判断。

5.6.3 CompareValidator 控件

CompareValidator 控件将用户输入到当前控件的值与输入到其他控件的值或常数值进行比较。除了确保值的正确性之外，CompareValidator 控件还具有保证输入的数字型、日期型数据格式正确的作用，其特有属性如表 5-6 所示。

表 5-6 CompareValidator 控件的特有属性

属性名称	说明
ControlToCompare	属性值为另一个控件的 ID，当前被验证的输入控件的值会与该控件的值进行比较
Operator	要执行的比较操作类型
Type	要比较的两个值的数据类型
ValueToCompare	属性值为一个常数值，当前被验证的输入控件的值会与该常数值进行比较

ControlToCompare 属性的优先级比 ValueToCompare 属性要高。如果同时设置了这两个属性，则 ControlToCompare 属性的值起作用。

比较操作类型（Operator 属性值）可以是以下值之一：

- DataTypeCheck：只对数据类型进行的比较。
- Equal（默认值）：相等。
- GreaterThan：大于。
- GreaterThanEqual：大于或等于。
- LessThan：小于。
- LessThanEqual：小于或等于。
- NotEqual：不等于。

Type 属性指定用于比较的数据类型，其值可以如下。

- String：字符串数据。
- Integer：32 位有符号整数数据。
- Double：双精度浮点数。
- Date：日期数据类型。
- Currency：货币数据类型。

为网站新增一个页面，名为CompareValidator.aspx，将Default.aspx页面中由<div>标记所包含的代码复制过来。

在"确认密码"输入框的后面增加一个CompareValidator控件，用于比较所输入的两个密码是否一致，代码如下：

```
< asp: CompareValidator ID="comPassword" runat="server" ErrorMessage="两个密码必须一致。"
ControlToCompare="Password1" ControlToValidate="Password2">
</asp: CompareValidator>
```

在"年龄"输入框的后面再增加一个CompareValidator控件，用于将所输入的值与一个常数值比较，代码如下：

```
<asp: CompareValidator ID="comAge" runat="server" ControlToValidate="Age" ErrorMessage="年龄必须
大于15岁。" Operator="GreaterThanEqual" Type="Integer" ValueToCompare="15">
</asp: CompareValidator>
```

上述两个CompareValidator控件一个用于比较两个输入控件的值，一个用于将输入值与一个常数值比较。将输入的年龄与常数值比较时，Type属性被设置为Integer，这时如果输入的值为非数字，验证也会报错。

5.6.4 RangeValidator 控件

RangeValidator控件验证输入值是否在指定范围内。该控件不仅能对数值进行验证，还可以对字符串、日期进行验证。RangeValidator控件的特有属性如表5-7所示。

表5-7 RangeValidator控件的特有属性

属性名称	说明
MaximumValue	验证范围的最大值
MinimumValue	验证范围的最小值
Type	要比较数值的数据类型

MaximumValue属性和MinimumValue属性用于设定验证范围，其指定值必须要能转换为Type属性所指定的数据类型，否则会引发异常。

为网站新增一个页面，名为RangeValidator.aspx，将Default.aspx页面中由<div>标记所包含的代码复制过来。

在"年龄"输入框的后面增加一个RangeValidator控件，用于限定所输入的年龄范围(在15～200之间)，代码如下：

```
<asp: RangeValidator ID="rangAge" runat="server" ErrorMessage="年龄必须在15~ 200之间。" Type="
Integer" MaximumValue="200" MinimumValue="15"
ControlToValidate="Age">
</asp: RangeValidator>
```

5.6.5 RegularExpressionValidator 控件

RegularExpressionValidator 是一个功能强大的验证控件，使用它可以确定所输入的值是否与某个正则表达式所定义的模式相匹配。

有关正则表达式的详细内容是一个复杂的话题，本书从略，只给出一个简单示例，希望读者从中对 RegularExpressionValidator 控件的强大功能及使用方便性有一点感受。

为网站新增一个页面，名为 RegularExpressionValidator. aspx，将 Default. aspx 页面中由<div>标记所包含的代码复制过来。

在 Email 输入框的后面增加一个 RegularExpressionValidator 控件，用于验证所输入电子邮件地址的格式。

在集成开发环境右下部的属性窗口中修改 RegularExpressionValidator 控件的相关属性，其中 ValidationExpression 为验证正则表达式，可以手工输入，也可以使用正则表达式编辑器进行编辑。

在设计模式下，当光标进入属性窗口的 ValidationExpression 属性项的输入框时，右侧会出现一个省略号按钮，单击它可以进入“正则表达式编辑器”对话框，如图 5-11 所示。在标准表达式列表中已经列出了常用的表达式格式，本例选择“Internet 电子邮件地址”即可。

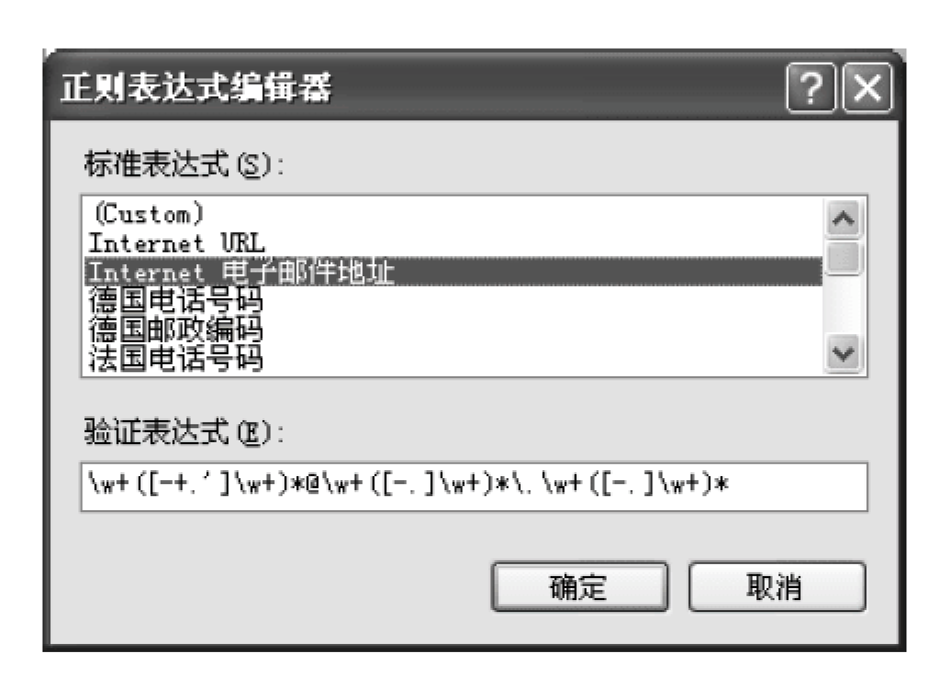


图 5-11 “正则表达式编辑器”对话框

修改属性后的控件代码如下：

```
<asp: RegularExpressionValidator ID= "revEmail" runat= "server" ErrorMessage=
"电子邮件地址格式错。" ControlToValidate= "Email"
ValidationExpression= "\w+ ([-+.']\w+)* @ \w+ ([-.]\w+)* \.\w+ ([-.]\w+)* ">
</asp: RegularExpressionValidator>
```

执行页面，如果输入的电子邮件地址格式有误，页面验证不通过，系统报错。

5.6.6 CustomValidator 控件

CustomValidator 控件允许用户使用自定义的验证程序来完成复杂逻辑的验证，这种验证可以在服务器端进行，也可以在客户端进行。

CustomValidator 控件是功能最强大的验证控件，也是使用方法最复杂、最灵活的验证控件。本小节不涉及该控件的细节，只给出一个小示例。

如果要在服务器端进行验证，需要为 CustomValidator 控件的 ServerValidate 事件编写处理函数；如果要在客户端进行验证，则需要在 CustomValidator 控件的 ClientValidationFunction 属性中指定客户端验证脚本的函数名称，本小节示例只演示了客户端验证的方法。

为网站新增一个页面，名为 CustomValidator.aspx，将 Default.aspx 页面中由<div>标记所包含的代码复制过来。

在"年龄"输入框的后面增加一个 CustomValidator 控件，用于限定所输入的年龄范围（在 15～200 之间），代码如下：

```
<asp:CustomValidator ID="cvAge" runat="server" ErrorMessage="年龄必须在 15~200 之间。"
ClientValidationFunction="ClientFunction" ControlToValidate="Age">
</asp:CustomValidator>
```

代码中指定由一个名为 ClientFunction 的客户端函数对输入年龄进行验证。

在页面的<head>标记之间增加 ClientFunction 函数的实现，代码如下：

```
<script type="text/javascript">
    function ClientFunction(source,args)
    {
        if ((args.Value>=15)&& (args.Value<=200))
            args.IsValid=True;
        else
            args.IsValid=False;
        return;
    }
</script>
```

用户输入的值由 args.Value 获得，在程序中进行验证，验证结果通过设置 args.IsValid 的值来返回。

习　　题

1. Calendar 控件主要提供哪些功能？

2. 创建一个 HTML 文档，在上面放置一个 Calendar 控件，尝试通过改变 Calendar 控件的属性来修改日历的外观形式。

3. 在习题 2 的基础上，参照 5.1.3 小节的介绍，通过对 Calendar 控件编程，实现如图 5.3 所示的执行效果。

4. 创建一个空白 HTML 文档，分别使用 FileUpload 控件和 HTML 上传控件<input id="File1" type="file" />来实现一个简单的文件上传功能。运行该文档，观察两种实现方法的执行效果。

5. 简述如何通过 FileUpload 控件的 PostedFile 属性对上传文件进行操作。

6. Wizard 控件的主要功能是什么？

7. 改变 Wizard 控件外观的简便方法是什么？

8. 参照 5.3 节的介绍，创建一个 HTML 文档，实现如图 5-6 所示的执行效果。

9. 简述 PlaceHolder 控件的作用。在应用中，使用 PlaceHolder 控件和 Panel 控件

有何区别?

10. 简述在哪些情况下可以使用 AdRotator 控件。

11. 描述 AdRotator 控件所使用的 XML 数据源文件的格式。

12. 简述验证控件的作用。ASP. NET 提供哪几类验证控件?简单描述各类验证控件的功能。

13. 简述如何使用 RequiredFieldValidator 控件对列表选择控件进行验证。

14. 简述如何通过程序对整个页面的验证结果进行判断和处理。

第 6 章 chapter 6

构 建 网 站

在前面的章节中主要介绍独立的控件，它们是构成网页的基本单位。而一个网站是由多个网页组成的，ASP.NET 有完备的机制将这些网页组成一个统一的整体——ASP.NET 网站。

本章主要介绍 ASP.NET 网站的组织及控制机制。

6.1 ASP.NET 网站综述

6.1.1 解决方案和项目

一个典型的 ASP.NET 网站由许多文件组成，这些文件包括 Web 窗体文件(.aspx)、源程序文件(.cs 或 .vb)、程序集(.exe 或 .dll)、图片(.jpg 或 .gif)等。VS2005 将这些文件统一组织在一个文件夹中，这个文件夹的所有内容组成一个 ASP.NET 网站，也称为一个 Web 应用程序。

当使用 VS2005 进行开发时，网站是"项目"的一种，称为网站项目。除网站外，使用 VS2005 还可以开发许多其他种类的项目，如 Windows 应用程序、类库、控制台应用程序等。但由于网站的应用比较多，且采用了与其他项目不同的组织方式，所以 VS2005 在创建和打开项目时都需要对网站项目进行单独的处理。本书介绍的主要就是网站项目的开发。

当新建一个网站(项目)时，VS2005 会自动为其创建一个解决方案，并显示在解决方案资源管理器中(一般情况下，解决方案资源管理器在集成开发环境的右部)。所谓解决方案就是将与一项开发任务相关的多个项目组织在一起。也就是说，一个解决方案可以包含多个网站项目和其他项目。例如创建一个称为 Wizard 的网站，在没有对其进行任何操作之前，其解决方案资源管理器界面可能如 6-1 所示。



图 6-1 新网站所包含的文件夹和文件

从图 6-1 可以看出，新创建的网站项目仅包括 ASP.NET 保留文件夹 App_Data 和一个默认网页 Default.aspx。但从中还可以看出项目的组织层次来：解决方案中可以包含多个项目，项目中又可以包含多个文件夹和文件项。

解决方案创建后系统会自动生成解决方案文件(.sln 和 .suo),为网站项目自动创建的解决方案文件默认存储在 My Documents\Visual Studio 2005\Projects 目录下与解决方案同名的子目录中。解决方案文件中存放了项目的配置信息,这样用 VS2005 再一次打开相同的解决方案时,就能够恢复到与上一次相同的操作状态。

6.1.2 ASP.NET 网站布局

网站项目由一系列文件组成。开发者可以为这些文件创建任意的目录结构,以方便开发。但是,为了更易于使用和管理网站,ASP.NET 保留了某些可用于特定类型内容的文件和文件夹名称。这些文件和文件夹被赋予特殊的含义和处理方法,本小节先简单介绍默认主页的概念和系统保留的应用程序文件夹。

1. 默认页

如果用户在请求的 URL 中只输入网站名而不指定特定页面,Web 服务器会为用户打开默认页(如果它存在的话)。使用默认页将使用户更容易定位自己开发的网站。

使用 VS2005 创建的网站,默认页为 Default.aspx,它保存在网站的根目录中。可以使用默认页作为网站的主页,或者在默认页中写入代码将用户请求重定向到真正的主页。例如,在应用实例的 Default.aspx 中就是根据是否具有 Cookie 对象和是否已经登录而判断使用哪个页面作为主页,然后重定向。

2. 应用程序文件夹

如果是一个新创建的网站,在“解决方案资源管理器”中的网站名称上单击右键,在弹出式菜单中选择“添加 ASP.NET 文件夹”,在子菜单中可以看到有 7 个文件夹可供选择。加上系统已经默认创建的 App_Data 文件夹,ASP.NET 规定这 8 个文件夹可用于存放特定类型的内容。除 App_Themes 文件夹外,其他的 ASP.NET 保留文件夹都不响应 Web 请求,但可以从应用程序代码进行访问。表 6-1 列出了 ASP.NET 保留的文件夹名称以及文件夹中通常包含的文件类型。

表 6-1 ASP.NET 保留文件夹

文件夹名称	说明
App_Browsers	包含 ASP.NET 用于标识特定浏览器并确定其功能的浏览器定义(.browser)文件
App_Code	包含希望作为应用程序一部分进行编译的实用工具类和业务对象(例如.cs、.vb 和 .jsl 文件)的源代码
App_Data	包含应用程序数据文件,如 MDF 文件、XML 文件和其他数据存储文件。ASP.NET 2.0 使用 App_Data 文件夹来存储应用程序的本地数据库,该数据库可用于维护成员资格和角色等信息
App_GlobalResources	包含编译到具有全局范围的程序集中的资源(.resx 和 .resources 文件)App_GlobalResources 文件夹中的资源是强类型的,可以通过编程方式进行访问

续表

文件夹名称	说　　明
App_LocalResources	包含与应用程序中的特定页、用户控件或母版页关联的资源(.resx 和 .resources 文件)
App_Themes	包含用于定义 ASP.NET 网页和控件外观的文件集合(.skin 和 .css 文件以及图像文件和一般资源)
App_WebReferences	包含在应用程序中使用的 Web 引用协议文件(.wsdl 文件)、架构文件(.xsd 文件)和发现文档文件(.disco 和 .discomap 文件)
Bin	包含要在应用程序中引用的控件、组件或其他代码的已编译程序集(.dll 文件)。在应用程序中可以自动引用 Bin 文件夹中的代码所表示的类

6.1.3　网站的组成文件

实际的网站要用到比图 6-1 所示多得多的文件才能完成所需的功能。图 6-2 是应用实例的网站结构,包含了多个用户自定义的文件夹(如 images、Uploads)和文件。

在图 6-2 所示的文件中,有两个文件需要特别说明。

图 6-2　一个实际的网站所包含的内容

1. Global.asax

Global.asax 文件是 ASP.NET 网站所拥有的一个全局性文件(文件名被 ASP.NET 所保留)。该文件中定义了应用程序的全局事件,它保存在应用程序的根文件夹中。

在"解决方案资源管理器"中的网站名称上单击右键,在弹出式菜单中选择"添加新项",在"模板"列表中选择"全局应用程序类",接受默认的文件名称,按"添加"按钮即可创建 Global.asax 文件。

新创建的 Global.asax 包括 5 个空的全局事件处理函数:Application_Start、Application_End、Application_Error、Session_Start 和 Session_End。这些事件的名称即可说明其含义,在后面的内容中也会用到其中一些事件。

2. Web.config

ASP.NET 网站的配置信息存储在 XML 文本文件中,名为 Web.config。Web.config 文件可以出现在 ASP.NET 应用程序的多个目录中。

为网站"添加新项",在"模板"列表中选择"Web 配置文件",接受默认的文件名称,按"添加"按钮,即可创建 Web.config 文件。

Web.config 的文件结构本书不详述,但在后面的章节中还会用到此文件。

新创建网站时，默认情况下只创建必需的文件和文件夹。因此，网站最初并未包含 Web. config 文件、Global. asax 文件等，也没有包含全部的 ASP. NET 保留文件夹。如果需要，用户可以自行创建。

6.1.4 网站文件类型

网站应用程序中可以包含很多文件类型，大多数 ASP. NET 支持的文件类型都可以使用“添加新项”菜单项自动生成。

前面的章节中已经涉及了一些文件类型，表 6-2 补充了一些文件类型并给出一个整体说明。

表 6-2 ASP. NET 支持的常用文件类型

文件类型	说明
. ascx	用户自定义的 Web 控件
. aspx	Web 页面文件，该文件可包含 Web 控件和其他业务逻辑
. browser	浏览器定义文件，用于标识客户端浏览器的启用功能
. cd	类关系图文件
. cs、. jsl、. vb	运行时要编译的类源代码文件。一般存放在 App_Code 子目录；但如果是 Web 内容文件的代码隐藏文件，则与其主文件位于同一目录
. dll	已编译的类库文件。一般存放在 Bin 子目录中。或者，可以将类的源代码放在 App_Code 子目录下
. master	母版页，它定义应用程序中引用母版页的其他网页的布局
. mdb、. ldb	Access 数据库文件。一般存放在 App_Data 子目录
. mdf	SQL 数据库文件。一般存放在 App_Data 子目录
. resources、. resx	资源文件，该文件包含指向图像、可本地化文本或其他数据的资源字符串。一般存放在 App_GlobalResources 或 App_LocalResources 子目录
. skin	用于确定显示格式的外观文件。一般存放在 App_Themes 子目录
. css	用于确定 HTML 元素格式的层叠样式表文件
. htm、. html	用 HTML 代码编写的静态 Web 文件

6.1.5 关于代码隐藏

如果用过 Visual Studio 的早期版本一定会记得：那时 HTML 标记、服务器端代码、客户端代码都是混排在一个文件(. asp)中的。这种代码组织形式对小的项目还是比较方便的，但对于大型的综合项目就会使代码难以控制，不符合软件工程的原则。

Visual Studio 2005 版虽然支持上述的单文件页模型，但其页面代码的默认编写方式已经改为代码隐藏模型，即将内容(表现)代码与源(逻辑)代码分开，将内容代码写在内容文件中，而将处理逻辑写在单独的代码隐藏文件中。

内容文件包括扩展名为 .aspx 的网页文件、扩展名为 .ascx 的用户自定义控件和扩展名为 .master 的母版页文件等。

代码隐藏文件则是根据所使用的语言，在内容文件名的基础上（不去掉原扩展名），再加上 .cs（使用 C＃语言）或 .vb（使用 Visual Basic 语言）作为扩展名。代码隐藏文件定义了从 Page 类派生的完整类。

先来看一下在创建一个新的网站时系统自动创建的默认主页 Default. aspx 和其代码隐藏文件 Default. aspx. cs 的原始内容。

以使用 C＃语言为例，Default. aspx 的内容如下：

```
<%@ Page Language="C#" AutoEventWireup="True"  CodeFile="Default.aspx.cs"
    Inherits="_Default" %>

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">

<html xmlns="http://www.w3.org/1999/xhtml" >
<head runat="server">
    <title>无标题页</title>
</head>
<body>
    <form id="form1" runat="server">
    <div>

    </div>
    </form>
</body>
</html>
```

说明：代码的第 1 行为一个@Page 指令，该指令定义网页一级属性。例如，在上述代码中，通过@Page 指令指定了 4 个网页级的属性，其中：

- Language 属性指明了默认的编程语言；
- AutoEventWireup 属性指明控件的事件是否自动匹配；
- CodeFile 属性指明了隐藏代码文件的文件名；
- Inherits 属性指明本页面编译后的类名称，该类在源代码文件中实现。

！DOCTYPE 指令用于指定文档类型定义（document type definition，DTD）。

上述代码的主体部分是 HTML 内容标签，其含义与一般的 HTML 文档相同，有关内容请参阅 1.1.1 小节。需要说明的是，ASP. NET 2.0 推荐，将文档内容（包括动态和静态内容）放在嵌套的<form>和<div>标签之内，便于引发页面回送和控制页面布局。

Default. aspx. cs 的内容如下：

```
using System;
using System.Data;
```

```
using System.Configuration;
using System.Web;
using System.Web.Security;
using System.Web.UI;
using System.Web.UI.WebControls;
using System.Web.UI.WebControls.WebParts;
using System.Web.UI.HtmlControls;

public partial class _Default : System.Web.UI.Page
{
    protected void Page_Load(object sender, EventArgs e)
    {

    }
}
```

说明：代码的开始是对一系列命名空间的引用。并不是所有的页面类都需要引用代码中列出的所有命名空间，但引用它们不会影响运行效率。

代码的主体部分声明了一个类，类名由页面内容文件的@Page 指令指定，该类从 Page 类派生。在对已有页面进行重命名时，代码中的上述内容可能需要手工修改。

6.1.6 网站的状态

用户在使用基于 Web 的应用程序时，所看到的是一个个独立的页面。用户在向服务器请求这些页面并得到响应时，使用的是 HTTP 协议，HTTP 协议是一种无状态的协议。但是，用户在不同的页面上进行不同的操作，这些操作有的是相互关联的。这就需要系统提供一些全局对象来保持页面之间的关联，将各页面组成一个有机的整体。这些对象包括 Request 对象、Response 对象、Application 对象、Session 对象和 Server 对象等。

另外，还需要对基于 Web 的应用程序做一些全局性的配置，这些配置一般是在文件 Global.asax 和 Web.config 中完成的。

本章后面的内容主要介绍这些全局对象和系统保留文件的使用方法。

6.2 Response 对象

Response 对象和 Request 对象是 ASP.NET 中非常重要的对象，用于在服务器端和客户端之间交互数据。Request 对象表示客户端向服务器发送的 HTTP 请求，Response 对象用于从服务器向客户端发送数据。

Response 对象用来控制发送给客户端的信息，包括直接发送信息给浏览器、重定向到另一个 URL 或设置 Cookie 等，Response 对象的常用属性和方法如表 6-3 所示。

表 6-3　Response 对象的常用属性和方法

属性名称	说　明
Buffer	当前页是否缓冲输出
BufferOutput	当前页是否缓冲输出
Charset	输出流的 HTTP 字符集
ContentType	输出流的 HTTP 内容类型
Cookies	Cookie 集合
Expires	在浏览器上缓冲存储的页面要多长时间过期。如果用户在页面过期之前“回退”到该页,则不再向服务器请求,而是显示缓存中的内容
ExpiresAbsolute	从缓存中移除缓存信息的绝对日期和时间
方法名称	**说　明**
Clear	清除缓冲区中的所有内容
End	将当前缓冲区中的所有内容发送到客户端,并停止该页的执行
Flush	向客户端发送当前缓冲区中的所有内容
Redirect	将客户端请求重定向到新的 URL
SetCookie	更新 Cookie 集合中的一个 Cookie
Write	将信息写入到 HTTP 响应输出流
WriteFile	将指定的文件直接写入 HTTP 响应输出流

下面对其中一些属性和方法做进一步的说明。

Charset 属性指定字符集名称,如:

```
Response.Charset="gb2312";
```

ContentType 属性指定响应的 HTTP 内容类型,如果未指定,其默认值为 text/HTML。

涉及 Cookie 的操作在本书应用实例的讲解中(第 1 章)作为一个专题进行讨论,本章不再介绍。

如果将 Buffer 属性设为 True,则当前页采用缓冲输出方式。如果缓冲输出,只有在当前页的所有服务器脚本都处理完毕,或调用了 Flush 或 End 方法后,服务器才将响应缓冲区的信息发送给客户端浏览器。将响应发送给客户端浏览器后,就不能再设置 Buffer 属性了。

从字面上看,BufferOutput 属性与 Buffer 属性的含义相同。Buffer 在以前的 ASP 版本中被提出并使用,目前已不再推荐,取而代之以 BufferOutput 属性。ASP.NET 2.0 使用 Buffer 属性只是为了向上兼容,在今后编程时,最好使用 BufferOutput。

Expires 属性指定了在浏览器上缓冲存储的页面要多长时间过期。在过期之前,用户在客户端用“回退”的方式再回到该页时,就会显示缓冲区中的页面,而不必再向服务器请求;如果已过期,则需要向服务器重新请求。例如,可将登录页面的 Expires 属性设

为0,这样该页面在登录后会立即过期,这也是一项安全措施。

ExpiresAbsolute 属性则指定在浏览器上缓冲存储的页面确切的到期日期和时间。在未到期之前,用户在客户端用"回退"的方式再回到该页时,就会显示缓冲区中的页面。

Clear 方法用于清除缓冲区中的所有 HTML 输出,可以在发生错误的情况下使用该方法。

注意: 只有将 BufferOutput 属性设为 True 时才能使用该方法。

End 方法将当前缓冲的内容发送到客户端,并停止该页的执行;调用 End 方法后,服务器会停止处理脚本并返回当前结果。Flush 方法也是将当前缓冲区的内容立即发送到客户端,所不同的是,调用过 Flush 方法后,页面还可以继续执行。

程序往往需要根据对用户请求的不同处理结果做出不同的响应。这样,Redirect 方法就是一个非常有用的方法,它可以将页面立即重定向到一个指定的 URL。在本书的应用实例中,就多处使用了 Redirect 方法进行输出重定向。

注意: 一旦使用了 Redirect 方法,任何在当前页中已经设置的响应内容都会被忽略。

Write 肯定是 Response 对象最常用的方法,它将指定的字符串写到当前的 HTTP 输出流。

来看一个简单示例。

创建一个名为 Response 的网站,打开默认主页的源程序文件 Default.aspx.cs,将 Page_Load 函数的内容修改为:

```
if (!Page.IsPostBack)
{     //输出 Respose 对象的属性值
    Response.Write("Buffer 属性: "+Response.Buffer.ToString()+"<br>");
    Response.Write("BufferOutput 属性: "+Response.BufferOutput.ToString()
        +"<br>");
    Response.Write("Charset 属性: "+Response.Charset.ToString()
        +"<br>");
    Response.Write("ContentType 属性: "+Response.ContentType.ToString()
        +"<br>");
    Response.Write("Cookies 属性: "+Response.Cookies.ToString()
        +"<br>");
    Response.Write("Expires 属性: "+Response.Expires.ToString()
        +"<br>");
    Response.Write("ExpiresAbsolute 属性: "
        +Response.ExpiresAbsolute.ToString()+"<br>");
}
//结束输出
Response.End();
```

执行该页面,结果如图 6-3 所示。

在浏览器端查看源文件,可以看到 HTML 文档中只包含由上述程序生成的内容,而不包括其他任何标记。

通过对输出结果的分析,可以帮助读者增强对 Response 对象各属性的理解。

Buffer属性:True
BufferOutput属性:True
Charset属性:utf-8
ContentType属性:text/html
Cookies属性:System.Web.HttpCookieCollection
Expires属性:0
ExpiresAbsolute属性:0001-1-1 0:00:00

图 6-3 使用 Response 对象控制输出

6.3 Request 对象

6.3.1 Request 对象概述

在服务器端可以使用 Request 对象访问基于 HTTP 请求传递的所有信息并进行处理，如用 POST 或 GET 方法传递的参数、Cookie 和客户端证书等。Request 对象的常用属性和方法如表 6-4 所示。

表 6-4 Request 对象的常用属性和方法

属性名称	说明
ApplicationPath	服务器上当前应用程序的虚拟路径
Cookies	客户端发送的 Cookie 的集合
FilePath	当前请求 URL 的虚拟路径
Form	窗体变量集合。可通过该属性访问窗体变量集合中的所有变量
Headers	HTTP 头集合
HttpMethod	客户端使用的 HTTP 数据传输方法(如 GET、POST 或 HEAD)
Params	QueryString、Form、ServerVariables 和 Cookies 项的组合集合
Path	当前请求的虚拟路径。Path＝FilePath＋PathInfo
PathInfo	URL 中的附加路径信息
PhysicalApplicationPath	当前正在执行的 Web 应用程序的根目录在服务器上的物理文件系统路径
PhysicalPath	与请求的 URL 相对应的服务器物理文件系统路径
QueryString	HTTP 查询字符串变量集合
RawUrl	当前请求的原始 URL
RequestType	客户端使用的 HTTP 数据传输方法(GET 或 POST)
ServerVariables	获取 Web 服务器变量的集合
TotalBytes	客户端在请求正文中包含的总字节数
Url	获取有关当前请求的 URL 的信息
UrlReferrer	获取有关客户端上次请求的 URL 的信息，该请求链接到当前的 URL
方法名称	**说明**
BinaryRead	对当前输入流进行指定字节数的二进制读取
MapPath	将当前请求的 URL 中的虚拟路径映射到服务器上的物理路径
SaveAs	将 HTTP 请求保存到磁盘
ValidateInput	验证由客户端浏览器提交的数据，如果存在具有潜在危险的数据，则引发一个异常

下面示例演示了表 6-4 中的大部分简单属性。

创建一个名为 Request 的网站，在默认主页 Default. aspx 上增加一个超链，代码如下：

```
<a href="DestPage.aspx?name=Gao Yi">导航到 DestPage.aspx</a>
```

为网站创建一个新的页面 DestPage. aspx。

打开其源程序文件 DestPage. aspx. cs，将 Page_Load 函数的内容修改为：

```
if (!Page.IsPostBack)
{     //输出 Request 对象的属性值
    Response.Write("ApplicationPath 属性："
        + Request.ApplicationPath.ToString() + "<br>");
    Response.Write("Cookies 属性："+ Request.Cookies.ToString() + "<br>");
    Response.Write("FilePath 属性："+ Request.FilePath.ToString()
        + "<br>");
    Response.Write("HttpMethod 属性："+ Request.HttpMethod.ToString()
        + "<br>");
    Response.Write("Path 属性："+ Request.Path.ToString() + "<br>");
    Response.Write("PathInfo 属性："+ Request.PathInfo.ToString()
        + "<br>");
    Response.Write("PhysicalApplicationPath 属性："+
        Request.PhysicalApplicationPath.ToString() + "<br>");
    Response.Write("PhysicalPath 属性："+ Request.PhysicalPath.ToString()
        + "<br>");
    Response.Write("QueryString 属性："+ Request.QueryString.ToString()
        + "<br>");
    Response.Write("RawUrl 属性："+ Request.RawUrl.ToString() + "<br>");
    Response.Write("RequestType 属性："+ Request.RequestType.ToString()
         + "<br>");
    Response.Write("Url 属性："+ Request.Url.ToString() + "<br>");
    Response.Write("UrlReferrer 属性："+ Request.UrlReferrer.ToString()
        + "<br>");
}
//结束输出
Response.End();
```

执行默认主页 Default. aspx，单击其中的超链，系统会导航到 DestPage. aspx 页面，显示如下内容：

```
ApplicationPath 属性：/Request
Cookies 属性：System.Web.HttpCookieCollection
FilePath 属性：/Request/DestPage.aspx
HttpMethod 属性：GET
Path 属性：/Request/DestPage.aspx
PathInfo 属性：
```

```
PhysicalApplicationPath属性：E:\NetworkAppCode\ch06\Request\
PhysicalPath属性：E:\NetworkAppCode\ch06\Request\DestPage.aspx
QueryString属性：name=Gao+Yi
RawUrl属性：/Request/DestPage.aspx?name=Gao%20Yi
RequestType属性：GET
Url属性：http://localhost:3835/Request/DestPage.aspx?name=Gao Yi
UrlReferrer属性：http://localhost:3835/Request/Default.aspx
```

同样，通过对输出结果的分析，可以帮助读者增强对Request对象常用属性的理解。

6.3.2 Params属性

Params是一个集合属性，它是其他几个集合属性的组合集合，包括QueryString、Form、ServerVariables和Cookies。与其他几个集合属性一样，其类型是NameValueCollection对象，可以通过键或索引访问集合中各项的字符串值。

在实际的编程中，Params属性经常被用来在服务器端提取客户端传来的参数值，使用C#语言取单个参数的方法如下：

```
Request.Params[参数名]
```

继续完成6.3.1小节的示例，演示使用Params属性的一般方法。

打开DestPage.aspx.cs，在Page_Load函数中增加如下内容：

```
//输出参数值
Response.Write("<br>输出参数值：<br>");
if (Request.Params["name"] != null)
    Response.Write("参数 name: "+Request.Params["name"].ToString()
        +"<br>");
else
    Response.Write("无参数 name。<br>");
if (Request.Params["birthday"] != null)
    Response.Write("参数 birthday: "
        +Request.Params["name"].ToString()+"<br>");
else
    Response.Write("无参数 birthday。<br>");
```

当系统再次执行并导航到DestPage.aspx页面时，显示结果增加了如下内容：

```
输出参数值：
参数 name: Gao Yi
无参数 birthday。
```

提示：Request对象还提供了一个BinaryRead方法，用于读取POST请求中的未加工二进制数据。这是一个底层功能，当使用其他属性、方法不能满足要求时，可使用BinaryRead方法。

6.3.3 ServerVariables 属性

在浏览器与服务器之间交互使用的是 HTTP 协议，在 HTTP 的标题文件中会记录一些客户端的信息，如 IP 地址、端口号等。有时在服务器端需要根据不同的客户端信息做出不同的响应，这时就需要使用 ServerVariables 属性获取所需要的信息。

ServerVariables 是 Web 服务器变量的集合，使用 C# 语言，取单独服务器变量的方法如下：

```
Request.ServerVariables[服务器变量名]
```

下面示例演示 ServerVariables 都包含哪些服务器变量，并取各变量的值。

为 Request 网站的默认主页 Default.aspx 再增加一个超链，代码如下：

```
<a href="ServerVariables.aspx?name=Gao Yi">导航到 ServerVariables.aspx</a>
```

为网站创建一个新的页面 ServerVariables.aspx。打开其源程序文件 ServerVariables.aspx.cs，将 Page_Load 函数的内容修改为：

```
if (!Page.IsPostBack)
{    //输出服务器环境变量值
    Response.Write("使用 ServerVariables 属性获得服务器环境变量："
        +"<br><br>");
    int i;
    //取得所有的键
    String[] arr1=Request.ServerVariables.AllKeys;
    for (i=0; i<arr1.Length; i++)
    {
        Response.Write("Key: "+arr1[i]+"<br>");
        Response.Write("Val: "+
            Request.ServerVariables[arr1[i]].ToString()+"<br><br>");
    }
}
//结束输出
Response.End();
```

执行 Default.aspx，单击相关的超链，导航到 ServerVariables.aspx 页面，会显示所有服务器变量的名称及值。下面是部分显示内容，从中可以分析出部分常用的服务器变量的含义（有的加上了必要的说明）：

```
Key: APPL_PHYSICAL_PATH
Val: E:\NetworkAppCode\ch2_05\Request\
Key: AUTH_TYPE
Val: NTLM
Key: AUTH_USER
Val: GYIBM\Administrator
Key: AUTH_PASSWORD
```

```
Val:
Key: LOGON_USER(客户的登录账号)
Val: GYIBM\Administrator
Key: CONTENT_LENGTH(客户端发出内容的长度)
Val: 0
Key: CONTENT_TYPE(客户端发出内容的类型)
Val:
Key: LOCAL_ADDR(服务器也许会有多个 IP地址,此变量返回接受请求的服务器地址)
Val: 127.0.0.1
Key: PATH_INFO
Val:/Request/ServerVariables.aspx
Key: PATH_TRANSLATED
Val: E: \NetworkAppCode\ch2_05\Request\ServerVariables.aspx
Key: REMOTE_ADDR(客户机 IP地址)
Val: 127.0.0.1
Key: REMOTE_HOST(客户机名称)
Val: 127.0.0.1
Key: REMOTE_PORT
Val:
Key: SERVER_NAME(出现在 URL中的服务器主机名、DNS化名或 IP地址)
Val: localhost
Key: SERVER_PORT(URL中请求的服务器端口号)
Val: 3835
Key: URL
Val:/Request/ServerVariables.aspx
Key: HTTP_HOST
Val: localhost: 3835
Key: HTTP_REFERER
Val: http://localhost: 3835/Request/Default.aspx
```

由上述输出内容可以看出,有些服务器变量通过 Request 对象的属性也能直接得到,直接使用 Request 对象的属性更方便一些。

6.4 Application 对象

ASP.NET 应用程序是单个 Web 服务器上的某个虚拟目录及其子目录范围内的所有文件、页、处理程序、模块和代码的总和。

Application 对象在某个应用程序的所有用户之间共享信息,并在服务器运行期间持久地保存数据。

Application 对象在第一次有客户端请求本应用程序的任何 URL 时创建。它存储在服务器的内存中,因此,与在数据库中存储和检索信息相比,对 Application 对象的操作执行速度更快。也正是由于它存储在内存中,Application 对象适合用于存储那些数量较少、不随用户数量而变化的常用数据。

可以将 Application 对象看成是应用程序全局变量的集合，使用 C# 语言可以用如下方法访问用户自定义的应用程序变量：

```
Application[应用程序变量名]
```

创建一个名为 Application 的网站。

在网站的默认主页 Default.aspx 上增加一个 Label 控件，其 ID 为 Label1。打开默认主页的源程序文件 Default.aspx.cs，为 Page_Load 函数增加如下代码(不包括行号)：

```
1:    Application.Lock();
2:    Application["PageSize"]="18";
3:    Application.UnLock();
4:    int ps=Convert.ToInt32(Application["PageSize"].ToString());
5:    Label1.Text=ps.ToString();
```

在上述代码中，为应用程序定义了一个应用程序变量，名为 PageSize(在很多实用的应用程序中，用类似的变量表示分页列表时每页的默认行数)。其中：

第 2 行代码写该变量。在写之前，该变量不必存在。

第 4 行代码读该变量。在读之前，所读的变量必须存在，否则会引发一个 System.NullReferenceException 异常。

Application 对象采用自由线程模式，即 Application 对象数据可由多个线程同时访问。因此，有时可能需要以线程安全的方式进行应用程序变量的更新。可以使用 Application 对象的 Lock 和 UnLock 方法来确保数据的完整性，如代码的第 1、第 3 行所示。

注意：由于 Application 对象存储在服务器内存中，因此每当停止或重新启动应用程序时，Application 对象都将丢失。例如，如果更改了 Web.config 文件，则要重新启动应用程序，所有应用程序状态都将丢失。

6.5 Session 对象

众所周知，HTTP 是一个无状态的协议。服务器在接收到客户端的请求后建立连接，在响应请求后断开连接。在服务器看来，每一次新的请求都是单独存在的，与以前的任何请求无关。因此，当用户在各页面之间跳转时，服务器根本无法知道并记录用户操作在各页面之间转换的过程及当前的操作状态。在早期基于 Web 的应用程序开发中，如何维持用户的请求状态总是应用程序开发的核心内容。

为了解决上述问题，各种 Web 服务器都进行了有针对性的增强，提出了各自的解决方案。ASP.NET 基于微软的 IIS Web 应用服务器，引入了 Session 对象，很好地解决了上述问题，给 Web 应用程序的开发带来了巨大的方便。

Session 对象存储特定的用户会话所需要的信息。当用户在应用程序的页之间跳转时，存在 Session 对象中的变量不会被清除，只要该用户还在访问应用程序的页面，这些变量就始终存在。

当用户请求来自某个应用程序的 Web 页时，如果该用户还没有会话，系统会自动为

其创建一个 Session 对象。当会话过期或被放弃后，服务器将终止该会话。

Session 对象与 Application 对象的本质区别在于：每个应用程序只有一个 Application 对象，被所有用户所共享；而每个应用程序可以有多个 Session 对象，应用程序的每个访问用户都有自己独享的一个 Session 对象。Session 对象的常用属性和方法如表 6-5 所示。

表 6-5　Session 对象的常用属性和方法

属性名称	说　　明
IsNewSession	当前会话是否是新会话（与当前请求一起创建）
SessionID	获取会话的唯一标识符
Timeout	在当前会话的各请求之间所允许的时间间隔（以分钟为单位，超过这个时间没有新的请求，则认为会话过期）
方法名称	**说　　明**
Abandon	取消当前会话
Clear	从会话状态集合中移除所有的键和值
Remove	删除会话状态集合中的项

Session 对象的应用示例参阅 6.7 节。

6.6　Server 对象

Server 对象提供了访问服务器对象的方法和属性（见表 6-6），可以获取服务器的信息，如应用程序路径等。

表 6-6　Server 对象的常用属性和方法

属性名称	说　　明
MachineName	服务器的计算机名称
ScriptTimeout	请求超时值（以秒为单位）
方法名称	**说　　明**
ClearError	清除前一个异常
GetLastError	返回前一个异常
HtmlDecode	对已编码（消除了无效 HTML 字符）的字符串进行解码
HtmlEncode	对要在浏览器中显示的字符串进行编码
MapPath	返回与 Web 服务器上的指定虚拟路径相对应的物理文件路径
UrlDecode	对字符串进行解码，该字符串为了进行 HTTP 传输而进行编码并在 URL 中发送到服务器
UrlEncode	编码字符串，以便通过 URL 从 Web 服务器到客户端进行可靠的 HTTP 传输

MapPath 方法在第 4 章关于 FileUpload 控件的示例中使用过,利用 Request 对象的某些属性也可以获得同样的数据。

在前述各章的示例中,经常会由程序控制在客户端(如一个 Label 控件上)显示一个字符串,这在大多数情况下不会出现问题。但如果在字符串中包含 HTML 标记则无法正常显示,这是因为客户端浏览器会对这些标记进行解释执行。

可使用 Server 对象的 HtmlEncode 方法将包含 HTML 标记的字符串编码为不包含 HTML 标记的字符串,这样在客户端经浏览器再次"解释"之后就可以按照用户意愿输出原来带有 HTML 标记的字符串了。HtmlDecode 方法是 HtmlEncode 方法的反操作。

创建一个名为 Server 的网站。

在网站的默认主页 Default. aspx 上增加一个 Label 控件,其 ID 为 Label1。打开默认主页的源程序文件 Default. aspx. cs,为 Page_Load()函数增加如下代码:

```
Label1.Text= "abc <strong>def</strong>ghi<br/>"
    + Server.HtmlEncode("abc <strong>def</strong>ghi<br/>");
```

执行该页面,其输出为:

```
abc def ghi
abc <strong>def</strong>ghi<br/>
```

可见,输出的第 1 行并不是原字符串,而是将原字符串按 HTML 语法进行"解释"之后的形式,即:其中的"def"被用粗体显示,串后加了一个换行。第 2 行显示的虽然是原字符串,但其中已经包含了先在服务器端编码,再在客户端由浏览器"解释"的过程。

当 URL 地址中包含非英文字符时,为了传输的安全性,应该对这些非英文字符进行编码,可以使用 UrlEncode 完成此工作。在接收到已编码的 URL 之后,可使用 UrlDecode 对其解码。

在许多高级应用的编程中,往往需要用到 ScriptTimeout 属性。

ScriptTimeout 属性指定程序脚本在服务器端可运行的最长时间,如果超过这个时间仍然没有完成,则会因超时而终止。系统会有一个 ScriptTimeout 的默认值,这个值随应用服务器版本的不同而有所不同。

设置 ScriptTimeout 是一项安全措施,可避免因运行错误代码(如死循环等)而长时间占用服务器资源,从而影响服务器效率,甚至造成服务器瘫痪。

但是,设置 ScriptTimeout 也可能造成正常代码的意外终止。如有些代码的执行可能比较耗时,如数据库操作等。这些代码执行的往往是关键操作,如果被意外终止,会给用户造成很不好的使用感受,甚至还会对系统产生破坏,这也是大家不希望看到的。

因此,在进行关键操作之前,可以根据代码执行的预期时间对 ScriptTimeout 属性进行设置。如:在海量数据库中进行多表交叉查询时,可以预先将 ScriptTimeout 设置为一个比较大的值。

下面通过示例介绍 ScriptTimeout 的读取和设置方法。

在页面上再增加两个 Label 控件,其 ID 分别为 Label2 和 Label3。在 Page_Load()函数中增加如下代码(不包括行号):

```
1:      Label2.Text=Server.ScriptTimeout.ToString();
2:      Server.ScriptTimeout=20;
3:      Label3.Text=Server.ScriptTimeout.ToString();
4:      //while (True)
5:      //{
6:      //    int i=100;
7:      //}
```

再次执行页面,增加了如下的输出:

```
110
20
```

Label2 上输出的是当前页面的默认 ScriptTimeout 值。第 2 行代码改变了 ScriptTimeout 属性的值,Label3 上输出的是改变后的值。

被注释掉的代码显然是一个死循环。

上述代码不具有实际意义,只是一个演示。如果去掉 4 至 6 行的注释,再次执行页面,则会看到“请求已超时”的出错提示。改变第 2 行的设置值,再次执行,会感受到超时时间的变化。

6.7 构建网站示例

本节创建一个名为 ApplicationAndSession 的网站,模拟在真实的应用程序中如何维护程序状态。示例虽然简单,其方法却被大多数实际的 ASP. NET 网站所采用。

为 ApplicationAndSession 网站创建一个全局应用程序类 Global. asax。如前文所述,该文件中已经包含了 5 个空的全局事件处理函数。并不是所有函数都必须重写,本例只重写其中两个。将其中相应部分代码改为:

```
void Application_Start(object sender, EventArgs e)
{
    //在应用程序启动时运行的代码
    Application["ApplicationName"]="畅想网络学院";
    Application["PageSize"]="18";
}
void Session_Start(object sender, EventArgs e)
{
    //在新会话启动时运行的代码
    Session["REMOTE_ADDR"]=
      Request.ServerVariables["REMOTE_ADDR"].ToString();
}
```

在 Application_Start 函数中设置了两个应用程序变量,这些变量在第一个用户请求本应用程序的页面时创建,所有用户可见。

在 Session_Start 中设置了一个会话变量 REMOTE_ADDR，记录当前客户机的 IP 地址。

为网站创建一个新的页面 Login.aspx，模拟一个登录界面，其内容代码相关部分为：

```
<table width="30% " border="0" cellpadding=0 cellspacing=0>
    <tr>
      <td align=right>用户名：</td>
      <td  align=left>
        <asp:TextBox ID="UserId" Runat="server" Width="120"/>
      </td>
    </tr>
    <tr>
      <td align=right>密码：</td>
      <td  align=left>
        <asp:TextBox ID="Password" Runat="server" Width="120"
            TextMode="Password"/>
      </td>
    </tr>
    <tr>
      <td colspan="2">
        <asp:Button ID="LoginBtn" Runat="server" Text="登录"
            Width="70px" OnClick="LoginBtn_Click"></asp:Button>
      </td>
    </tr>
</table>
```

在其隐藏代码文件中，LoginBtn_Click 事件处理函数的代码为：

```
protected void LoginBtn_Click(object sender, EventArgs e)
{
    //应该先判断用户信息是否合法,本例从略
    Session["UserId"]=Server.HtmlEncode(UserId.Text.Trim());
    Session["Password"]=Server.HtmlEncode(Password.Text.Trim());
    Response.Redirect("MainPage.aspx");
}
```

再为网站创建一个新的页面 MainPage.aspx，模拟应用程序主页，其内容代码相关部分为：

```
应用程序信息：<br/>
<asp:Label ID="Label1" runat="server"></asp:Label><br/>
<br/>
会话信息：<br/>
<asp:Label ID="Label2" runat="server"></asp:Label><br/>
<asp:Label ID="Label3" runat="server"></asp:Label></div>
```

在其隐藏代码文件中,Page_Load()函数的代码为:

```
if (Application["ApplicationName"] != null)
    Label1.Text= "系统名称:"+ Application["ApplicationName"].ToString()
    + "<br> "+ "默认每页的行数:"+ Application["PageSize"].ToString();
if (Session["REMOTE_ADDR"] != null)
    Label2.Text= "客户机地址:"+ Session["REMOTE_ADDR"].ToString();
if (Session["UserID"] != null)
    Label3.Text= "当前用户:"+ Session["UserID"].ToString()+ "/"+
    Session["Password"].ToString();
```

运行页面 Login.aspx,输入任意的用户名(如 gao)和密码(如 yi)。单击登录按钮,MainPage.aspx 页面被加载,并显示如下内容:

```
应用程序信息:
系统名称:畅想网络学院
默认每页的行数:18

会话信息:
客户机地址:127.0.0.1
当前用户:gao/yi
```

注意:如果在开始时不执行 Login.aspx,而直接执行 MainPage.aspx,则"当前用户"一行不显示。说明无论从哪个页面开始执行,Global.asax 中的函数都会被执行。

习　　题

1. 一个典型的 ASP.NET 网站通常由哪些项组成?
2. 在一个 ASP.NET 网站中,什么是主页?什么是默认页?二者之间有何关联?
3. ASP.NET 保留文件夹有哪些?各保留文件夹中通常包含什么类型的文件?
4. Global.asax 文件在 ASP.NET 网站中有什么作用?
5. Web.config 文件在 ASP.NET 网站中有什么作用?该文件的内容是以什么格式存储的?
6. ASP.NET 网站通常包含哪些文件类型?各种不同类型的文件都有哪些用途?
7. 代码隐藏有何意义?是如何实现的?
8. 在 Web 应用中,使用哪些对象来保存网站的状态?
9. Response 对象有什么作用?
10. 简述 Response 对象的 Expires 和 ExpiresAbsolute 属性的含义。
11. 参照 6.2 节的介绍,创建一个名为 Response 的网站,在默认主页的源程序文件 Default.aspx.cs 中,对 Response 对象的常用属性和方法进行操作,观察程序执行效果。
12. 运行上题所创建的网站,在浏览器端查看源文件。将程序中的 Response.End

();语句注释掉,再次执行,再次在浏览器端查看源文件。分析这两次查看所得到的结果有何不同。

13. Request 对象有什么作用?

14. 参照 6.3 节的介绍,创建一个名为 Request 的网站,在默认主页的源程序文件 Default.aspx.cs 中,对 Request 对象的常用属性和方法进行操作,观察程序执行效果。

15. Application 对象有什么作用和特点?

16. Session 对象有哪些用途? Session 对象与 Application 对象有什么本质区别?

17. Server 对象有什么作用?

18. 简述 Server 对象的 ScriptTimeout 属性的作用。

19. 使用 VS2005 集成开发环境,自己动手创建 6.7 节介绍的 ApplicationAndSession 网站,并观察网站的执行效果。

第 7 章 chapter 7

应用 ADO. NET 编程

前面已经介绍了构建网站相关的基本知识。为了使网站能够提供高级的服务功能，绝大多数应用程序都需要具有对数据库中大量业务数据进行动态管理的能力。

ADO. NET 为 ASP. NET 提供高效的数据访问机制，同时能够和 XML 进行无缝集成。本章介绍应用 ADO. NET 进行编程的基本技巧。

从本章起，大部分与数据库操作有关的实例都是在 SQL Server 数据库上完成的。如果要运行这些实例，需要参照本书第 11 章的内容，在 SQL Server 上建立示例数据库。在本书以下的内容中，都假设示例数据库已经建立，访问用户为 sa，口令为 abc，数据库为 NetSchool。

7.1 ADO.NET 概述

ADO. NET 提供了对所有能够通过 OLE DB 对数据库进行访问的通用方法。通过 ADO. NET 可以连接到所有 OLE DB 支持的数据源，并对其中数据进行检索与更新。

ADO. NET 提供连接式和非连接式两种数据访问模式。

非连接式的数据访问主要使用 DataSet 对象。使用 DataSet 对象不一定非要与数据库相连接，但一般情况下，是把 DataSet 对象作为数据库(或部分数据库或来自多个数据源的数据)在内存中的一个副本来使用。程序可以像直接操作数据库中的数据一样操作 DataSet 对象中的数据。

连接式的数据访问主要使用 DataReader 对象。当需要处理大量数据时，一次性地将所有数据导入到内存再进行处理并不是一个好的方法；使用 DataReader 对象必须用连接的方式来访问数据库，一次只从数据库中取得必要的数据进行处理，处理完后，再从数据库中继续读入需要的数据。使用 DataReader 对象采用的是一种只读的、向前的、快速的数据库读取机制，这样可以提高应用程序的执行效率。

使用 ADO. NET 进行编程主要包括：使用 Connection 对象来连接数据库；使用 Command 对象执行数据库命令来操作数据库或使用 DataReader 对象来读取数据；使用 DataAdapter 对象将数据库中的数据填充到 DataSet 对象中，对 DataSet 对象中的数据进行操作等，如图 7-1 所示。

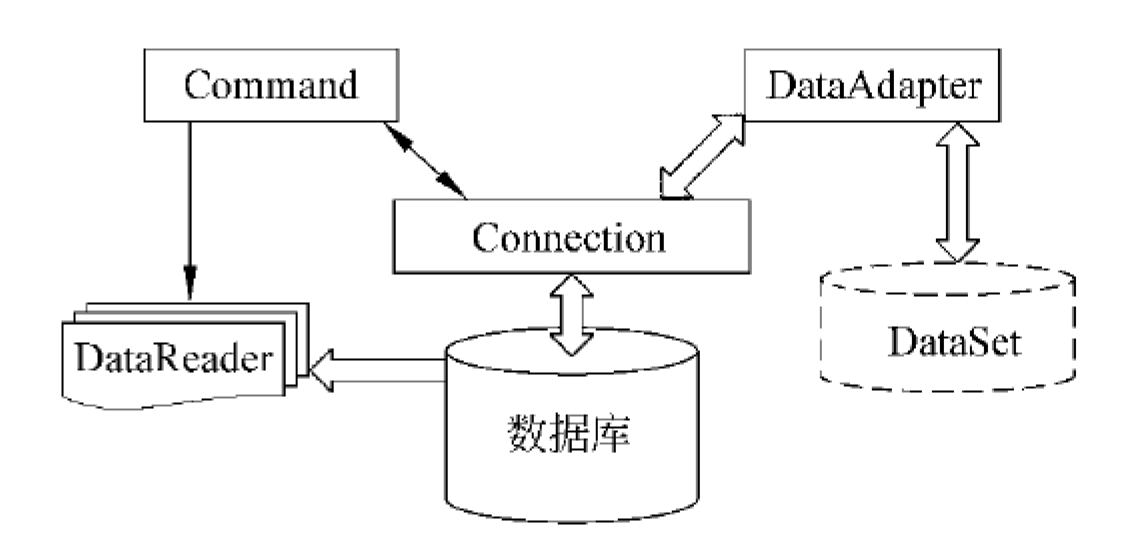


图 7-1 ADO.NET 对象模型

7.2 使用 ADO.NET 连接数据库

ADO.NET 使用 Connection 对象实现连接数据库的功能，它是操作数据库的基础，表示应用程序与数据库之间的唯一会话。

ADO.NET 支持以多种方式连接到数据库，例如，通过 ADO.NET 可以用通用的方式连接到所有 OLE DB 和 ODBC 支持的数据库。在.NET Framework 中，称处理数据库操作的底层支持程序为托管提供程序。除了上述通用方式这外，ADO.NET 还为 SQL Server 和 Oracle 这样的常用大型数据库提供了专用的托管提供程序，以便提高数据库的访问性能。

本节对上述通过 ADO.NET 连接数据库的常用方法分别进行概要介绍。

7.2.1 连接 SQL Server 数据库

通过 ADO.NET 连接 SQL Server 数据库一般是使用.NET Framework 提供的专用托管提供程序，其一般步骤如下。

1. 引用 System.Data.SqlClient 命名空间

使用 System.Data.SqlClient 命名空间可以开发对 Microsoft SQL Server 7.0 版或更新版本中的数据库进行访问的应用程序。System.Data.SqlClient 是.NET Framework 提供的 SQL Server 专用托管提供程序访问接口。使用时在程序的前部加上如下语句：

```
using System.Data.SqlClient;
```

2. 定义连接字符串

连接字符串中指明了 SQL Server 数据库的各种连接属性，以“键＝值”对的形式组合而成。“键”为连接属性名称，“值”为属性的取值，“键＝值”对之间以分号(;)隔开。

能够出现在 SQL Server 连接字符串中的常用属性如表 7-1 所示，注意，这些只是部分常用属性，全部属性的详细介绍请参考 MSDN 的相关内容。

表 7-1 能够出现在 SQL Server 连接字符串中的常用属性

属性名称	说明
Connect Timeout 或 Connection Timeout	等待与服务器连接的时间长度(以秒为单位),超过这个时间后,程序将停止等待并产生错误。默认为 15s
Data Source 或 Server 或 Address 或 Addr	要连接的 SQL Server 实例的名称或网络地址。可以在服务器名称之后指定端口号,其形式如: server=tcp: servername, portnumber 如果要连接的是本地数据库实例,servername 部分可用(local)表示。若要强制使用某个协议,则添加前缀 np、tcp 或 lpc
Initial Catalog 或 Database	SQL Server 数据库的名称
Integrated Security 或 Trusted_Connect	当为 False 时,将在连接中指定用户 ID 和密码;当为 True 时,将使用当前的 Windows 账户信息进行身份验证。默认为 False
Password 或 Pwd	SQL Server 账户登录的密码。出于安全性的考虑,微软建议不要使用此属性,建议使用 Integrated Security 属性
User ID	SQL Server 登录账户。出于安全性的考虑,微软建议不要使用此属性,建议使用 Integrated Security 属性

下面是一个 SQL Server 连接字符串的实例:

```
myConnectionString = "Data Source= (local); Initial Catalog= Northwind;
User ID= sa;Password= abc"
```

3. 创建 SqlConnection 对象

程序中用 SqlConnection 对象表示到 SQL Server 的一个连接。一般情况下,SqlConnection 对象的创建代码如下:

```
SqlConnection mySQLConnection= new SqlConnection(myConnectionString);
```

SqlConnection 对象有一些有用的公共属性,本节不再详细介绍,请参考 7.2.4 小节的实例。在上述创建连接对象的代码中,SqlConnection 对象的属性包含在连接字符串中,作为参数传入。在创建时也可以不指定连接字符串,在打开连接之前,再指定 SqlConnection 对象的 ConnectionString 属性。

4. 打开连接

```
mySQLConnection.Open();
```

5. 关闭连接

所有的数据库操作完成之后,应该关闭连接。

```
mySQLConnection.Close();
```

7.2.2 连接Oracle数据库

.NET也提供连接Oracle数据库的专用托管程序。连接Oracle数据库的步骤与连接SQL Server相同,因此,只介绍一些细节上的不同之处。

1. 引用System.Data.OracleClient命名空间

需要注意的是,System.Data.OracleClient不是.NET的默认数据库处理程序。因此,要使用System.Data.OracleClient,必须手工增加对它的动态链接库的引用,方法如下:

选择VS2005集成开发环境的"网站-启动选项"菜单项,在左侧的列表中选择"引用",单击"添加引用"按钮,在"添加引用"对话框中选中System.Data.OracleClient,单击"确定"按钮。

使用时在程序的前部加上如下语句:

```
using System.Data.OracleClient;
```

2. 定义连接字符串

与连接SQL Server数据库相同,连接Oracle数据库的连接字符串也采用"键=值"对的形式。各连接属性的定义不再详述,下面仅给出一个Oracle连接字符串的实例:

```
myConnectionString = "data source= orcl;user id= scott;password= tiger"
```

3. 创建OracleConnection对象

```
OracleConnection myOracleConnection= new OracleConnection(myConnectionString);
```

4. 打开连接

```
myOracleConnection.Open();
```

5. 关闭连接

```
myOracleConnection.Close();
```

7.2.3 通过OLE DB连接数据库

通过.NET提供的OLE DB托管提供程序,可以用同样的方法连接SQL Server、Oracle、Access和Excel等多种数据源。连接步骤与连接SQL Server相同,因此,只介绍一些细节上的不同之处。

1. 引用System.Data.OleDb命名空间

```
using System.Data.OleDb;
```

2. 定义连接字符串

下面是一个连接 Access 数据库的连接字符串实例：

```
myConnectionString = "Provider= Microsoft.Jet.OLEDB.4.0;
Data Source= E: \DB\AccessTest.mdb"
```

3. 创建 OleDbConnection 对象

```
OleDbConnection myAccessConnection= new OleDbConnection(myConnectionString);
```

4. 打开连接

```
myAccessConnection.Open();
```

5. 关闭连接

```
myAccessConnection.Close();
```

7.2.4 连接数据库实例

创建一个名为 ConnectToDatabase 的网站。网站功能为：主页第一次加载时显示一个下拉式列表和一个按钮，如 7-2 所示。

列表中有三个选项，分别为 SQL Server、Oracle 和 Access。选择一种数据库，单击“连接”按钮，系统尝试连接该数据库，并在页面再次加载时显示连接结果信息。

SQL Server
连接

图 7-2　连接数据库实例

第一步，在实际的应用程序中，很少像上文那样将连接字符串直接写在程序中，这样会降低程序的灵活性。通常的方法是在 Web. config 文件中定义连接字符串，然后在程序中读取。

为网站创建并打开 Web 配置文件 Web. config。从 Web. config 的初始代码可以看出，Web. config 是一个 XML 格式的文本文件。

将其中的<connectionStrings/>段改为：

```
<connectionStrings>
    <add name= "SQLConnectionString" connectionString= "Data Source=
(local); Initial Catalog= Northwind;User ID= sa;Password= abc"/>
    <add name= "OracleConnectionString" connectionString=
"data source= orcl;user id= scott;password= tiger"/>
    <add name= "AccessConnectionString"
connectionString= "Provider= Microsoft.Jet.OLEDB.4.0;Data Source=
E: \NetworkAppCode\ch07\ConnectToDatabase\App_Data\AccessTest.mdb"/>
</connectionStrings>
```

说明：分别定义了连接3种数据库的3个连接字符串。

提示：在实际的运行环境中，可能需要针对具体环境对其中的某些属性值进行修改。

第二步，在主页上增加一个下拉式列表控件和一个按钮控件，对其属性进行修改后，代码如下：

```
<asp:DropDownList ID="DropDownList1" runat="server">
    <asp:ListItem>SQL Server</asp:ListItem>
    <asp:ListItem>Oracle</asp:ListItem>
    <asp:ListItem>Access</asp:ListItem>
</asp:DropDownList><br/>
<asp:Button ID="Button1" runat="server" Text="连接"
    OnClick="Button1_Click"/> </div>
```

第三步，在 Default.aspx.cs 的开头加上对下列命名空间的引用，代码如下：

```
using System.Data.SqlClient;      //为了连接 SQL Server 数据库
using System.Data.OracleClient;   //为了连接 Oracle 数据库
using System.Data.OleDb;          //为了连接 OleDb 数据库
```

并用 7.2.2 小节所述的方法，增加对 System.Data.OracleClient 动态链接库的引用。

第四步，为按钮的单击事件处理函数增加代码如下：

```
string connectionString="";

try
{
    switch (DropDownList1.Text.ToString())
    {
        case "SQL Server":
            connectionString=ConfigurationManager.
            ConnectionStrings["SQLConnectionString"].ConnectionString;
            SqlConnection SQLConn=new SqlConnection(connectionString);
            SQLConn.Open();
            Response.Write("链接状态："+(SQLConn.State ==
                ConnectionState.Open ? "链接成功。" : "链接失败。")+"<br>");
            Response.Write("链接字符串："+SQLConn.ConnectionString+"<br>");
            Response.Write("ConnectionTimeout: "+
                SQLConn.ConnectionTimeout.ToString()+"<br>");
            Response.Write("DataSource: "+SQLConn.DataSource+"<br>");
            Response.Write("ServerVersion: "+SQLConn.ServerVersion+"<br>");
            SQLConn.Close();
            break;
        case "Oracle":
            connectionString=ConfigurationManager.
            ConnectionStrings["OracleConnectionString"].ConnectionString;
            OracleConnection OracleConn=
```

```
                        new OracleConnection(connectionString);
                    OracleConn.Open();
                    Response.Write("链接状态："+ (OracleConn.State ==
                        ConnectionState.Open ? "链接成功。" : "链接失败。")+ "<br>");
                    Response.Write("链接字符串："+ OracleConn.ConnectionString+ "<br>");
                    Response.Write("ConnectionTimeout: "+
                        OracleConn.ConnectionTimeout.ToString()+ "<br>");
                    Response.Write("DataSource: "+ OracleConn.DataSource+ "<br>");
                    Response.Write("ServerVersion: "+ OracleConn.ServerVersion
                        + "<br>");
                    OracleConn.Close();
                    break;
                case "Access":
                    connectionString= ConfigurationManager.
        ConnectionStrings["AccessConnectionString"].ConnectionString;
                    OleDbConnection AccessConn= new OleDbConnection(connectionString);
                    AccessConn.Open();
                    Response.Write("链接状态："+ (AccessConn.State ==
                        ConnectionState.Open ? "链接成功。" : "链接失败。")+ "<br>");
                    Response.Write("链接字符串："+ AccessConn.ConnectionString+ "<br>");
                    Response.Write("ConnectionTimeout: "+
                        AccessConn.ConnectionTimeout.ToString()+ "<br>");
                    Response.Write("DataSource: "+ AccessConn.DataSource+ "<br>");
                    Response.Write("ServerVersion: "+ AccessConn.ServerVersion
                        + "<br>");
                    AccessConn.Close();
                    break;
                default :
                    Response.Write("Please select a kind of Database.<br>");
                    break;
            }
        }
        catch (Exception ex)
        {
            ///显示连接错误的消息
            Response.Write(ex.Message+ "<br>");
        }
```

其中 ConfigurationManager 类在 System.Configuration 命名空间中定义，用于提供对配置文件中配置项的访问。

在上述代码中，根据所选择的数据库类型：

- 从 Web 配置文件中取相应的连接串。
- 创建连接对象。
- 打开连接。

- 取连接对象的各属性。
- 关闭连接。

如果连接失败，系统会抛出异常，并在异常处理中显示连接出错的相关信息。

第五步，执行网页，选择数据库，单击“连接”按钮。

如果连接 SQL Server 成功会显示类似下文的信息：

```
链接状态：链接成功。
链接字符串：Data Source=(local); Initial Catalog=Northwind;User ID=sa;
ConnectionTimeout: 15
DataSource: (local)
ServerVersion: 08.00.0194
```

如果连接 Oracle 成功会显示类似下文的信息：

```
链接状态：链接成功。
链接字符串：data source=orcl;user id=scott;
ConnectionTimeout: 0
DataSource: orcl
ServerVersion: 9.2.0.1.0 Oracle9i Enterprise Edition Release 9.2.0.1.0 - Production With the
Partitioning, OLAP and Oracle Data Mining options JServer Release 9.2.0.1.0- Production
```

如果连接 Access 成功会显示类似下文的信息：

```
链接状态：链接成功。
链接字符串：Provider=Microsoft.Jet.OLEDB.4.0;Data Source=
E: \NetworkAppCode\ch07\ConnectToDatabase\App_Data\AccessTest.mdb
ConnectionTimeout: 15
DataSource:
E: \NetworkAppCode\ch07\ConnectToDatabase\App_Data\AccessTest.mdb
ServerVersion: 04.00.0000
```

7.3 使用 Command 对象和 DataReader 对象

成功连接数据库之后，就可以对数据库进行查询和修改操作了。

对数据库进行查询，一个常用的方法就是使用 Command 对象和 DataReader 对象相结合读取数据。

从本节开始，数据库的操作都以 SQL Server 为例进行介绍，使用 SQL Server 专用托管提供程序（System.Data.SqlClient）。相应地，在读取数据时，也使用 System.Data.SqlClient 命名空间中的对象 SqlCommand 和 SqlDataReader。

对于 7.2 节介绍的其他数据库，需要使用与其托管提供程序相对应的 Command 对象和 DataReader 对象，使用方法基本相同，不再重复。

SqlCommand 对象表示要对 SQL Server 数据库执行的一个 Transact-SQL 语句或存储过程，使用 SqlCommand 对象可对 SQL Server 数据库进行查询、插入、修改和删除等

操作。SqlCommand 对象的常用属性和方法如表 7-2 所示。

表 7-2 SqlCommand 对象的常用属性和方法

属性名称	说　明
CommandText	要对数据源执行的 Transact-SQL 语句或存储过程
CommandTimeout	在终止执行命令的尝试并生成错误之前的等待时间
CommandType	SqlCommand 对象的执行方式
Connection	当前 SqlCommand 实例所使用的 SqlConnection 对象
Transaction	与当前 SqlCommand 相关联的 SqlTransaction 对象
方法名称	**说　明**
Cancel	尝试取消 SqlCommand 的执行
ExecuteNonQuery	执行 Transact-SQL 语句并返回受影响的行数
ExecuteReader	执行由 CommandText 所指明的 SQL 语句(一般是查询语句),将结果作为一个 SqlDataReader 对象返回
ExecuteScalar	执行查询,并返回查询结果集中第一行的第一列。忽略其他列或行
ExecuteXmlReader	执行由 CommandText 所指明的 SQL 语句(一般是查询语句),将结果作为一个 XmlReader 对象返回

CommandType 属性指明 SqlCommand 对象的执行方式,有以下三个可选值。

- StoredProcedure:需要将 CommandText 属性设为要执行的存储过程的名称。
- TableDirect:需要将 CommandText 属性设为要访问的表的名称,执行后返回该表的所有行和列。
- Text(默认值):需要将 CommandText 属性设为 SQL 文本命令。

如果有多项数据库操作需要作为一个整体来执行(或者全部执行,或者一项也不执行),则需要用到事务处理。数据库中"事务(Transaction)"的概念本书不再赘述,在 ADO. NET 中如果需要进行事务处理,其一般方法(忽略细节)是:

- SqlCommand 的每次执行为一项数据库操作。
- 将需要作为一个事务处理的各项数据库操作与一个 Transaction 对象相关联。

与 Transaction 对象相关联的方法是设置 SqlCommand 对象的 Transaction 属性。

调用 Cancel 方法尝试取消正在进行的数据库操作。之所以说"尝试",是因为取消操作不一定能成功,有些操作不能被取消。如果取消尝试失败,也不会抛出异常。如果没有要取消的内容,则什么都不做。

调用 ExecuteNonQuery 方法,既可以执行任何数据库 DDL 语句(如创建表、视图等),以完成对数据库结构的修改;也可以执行任何非查询 DML 语句(UPDATE、INSERT 或 DELETE),修改数据库中的数据。如果执行的是 UPDATE、INSERT 或 DELETE 语句,返回值为执行该命令所影响的行数;如果执行其他类型的语句,返回值为 —1。

要想执行查询语句并返回结果集,使用 ExecuteReader 方法或 ExecuteXmlReader

方法。但有时如果已知所执行的查询语句仅返回一个值，调用 ExecuteScalar 方法更方便。

SqlDataReader 对象提供一种对数据只读的、向前的数据读取方式。

- 只能通过 SqlDataReader 对象读取数据，不能通过它修改数据库。
- 一条记录读过之后，不能再重读一次。

SqlDataReader 对象的常用属性和方法如表 7-3 所示。

表 7-3 SqlDataReader 对象的常用属性和方法

属性名称	说明
FieldCount	列数
HasRows	当前 SqlDataReader 中是否包含数据记录
IsClosed	当前 SqlDataReader 是否已关闭
Item	根据列序号或列名称（以参数方式传递），取得 SqlDataReader 中当前行指定列的值
VisibleFieldCount	SqlDataReader 中未隐藏的字段的数目
方法名称	**说明**
Close	关闭 SqlDataReader 对象
GetBoolean、GetChar、GetFloat、GetInt32 等	获取指定列的指定类型的值，参数为从 0 开始的列序号
GetValue	获取指定列的值，参数为从 0 开始的列序号，返回值为 Object 类型
GetValues	获取当前行的所有列，将结果复制到 Object 数组中。返回值为数组中 Object 的数目
IsDBNull	返回某列的值是否为空，参数为从 0 开始的列序号
Read	使 SqlDataReader 前进到下一条记录

执行 SELECT 语句返回的 SqlDataReader 对象中包含一定数量的列（字段）。这些列有些是可见的，一般是在 SELECT 语句的查询字段列表中的列；有些则是隐藏的，如对于多字段主键，在部分主键字段上使用 SELECT 语句，则会将主键的其他字段作为隐藏字段返回。隐藏字段总是附加在可见字段后面。FieldCount 属性值为所有列的数量，而 VisibleFieldCount 属性值则仅为可见列的数量。

利用上述两个对象编程，常用的方法是：先创建一个执行查询功能的 SqlCommand 对象，再执行其 ExecuteReader 方法，就会将查询结果以一个 SqlDataReader 对象返回。请看下面的实例。

创建一个名为 UseADONET 的网站。网站的功能为：主页面上包含一个 DropDownList 控件；页面加载时从数据库中读取课程数据，并将这些数据添加到 DropDownList 控件的 Items 集合中列表显示。

第一步，创建 Web.config 文件，并用前节介绍的方法增加一个连接字符串如下：

```
<add name="SQLConnectionString" connectionString="Data Source=(local); Initial Catalog=NetSchool;
```

```
User ID=sa;Password=abc"/>
```

注意：这里使用了本书的示例数据库 NetSchool。

第二步，在主页上增加一行文字和一个 DropDownList 控件，代码如下：

```
课程列表：<br/>
<asp:DropDownList ID="DropDownList1" runat="server">
</asp:DropDownList>
```

第三步，在 Default.aspx.cs 的开头加上对 System.Data.SqlClient 命名空间的引用。为_Default 类增加一个私有成员：

```
private string connectionString=
ConfigurationManager.ConnectionStrings["SQLConnectionString"].ConnectionString;
```

将 Page_Load()函数改为：

```
protected void Page_Load(object sender, EventArgs e)
{
    if (!Page.IsPostBack)
    {
        FillData();
    }
}
```

实现 FillData()函数如下：

```
    private void FillData()
    {
        SqlConnection conn=new SqlConnection(connectionString);

        string cmdText="SELECT * FROM CLASS";
        SqlCommand command=new SqlCommand(cmdText, conn);

        try
        {
            //打开连接
            conn.Open();

            //执行查询
            SqlDataReader dr=command.ExecuteReader();
            while (dr.Read())
            {
                //向列表中添加 Item项
                DropDownList1.Items.Add(new ListItem(
                    dr["CLASSID"]+"-"+dr["CLASSNAME"].ToString(),
                    dr["CLASSID"].ToString()));
```

```
12:                 }
13:                 dr.Close();
            }
            catch (SqlException sqlex)
            {
                //显示错误信息
14:             Response.Write(sqlex.Message+"<br>");
            }
            finally
            {
                //关闭数据库链接
15:             conn.Close();
            }
        }
```

在上述代码中：

01 语句创建了一个连接对象 conn。

03 语句创建了一个命令对象 command。在创建命令对象时传入了两个参数，字符串 cmdText 是要执行的 SQL 语句，conn 指明操作是对 conn 所连接的数据库进行。

05 语句调用 command 的 ExecuteReader 方法，将结果返回给一个 SqlDataReader 对象 dr。

06 语句～12 语句为使用 SqlDataReader 对象的最常用方法：循环调用 SqlDataReader 对象的 Read 方法遍历 SqlDataReader 对象的所有行，利用字段名取指定列的值。

如果数据库操作过程中出现异常，14 句显示错误信息。

无论如何，代码的第 15 句总会被执行，数据库连接总会关闭。

第四步，执行页面，显示课程列表如图 7-3 所示。可以打开数据库中的 CLASS 表，与下拉式列表内容对照，看是否一致。

课程列表:
J01 - C语言程序设计

图 7-3　显示课程列表

7.4　使用 DataAdapter 对象和 DataSet 对象

DataSet 对象是 ADO. NET 中最复杂、功能最强的一个对象。DataSet 是数据库的内存驻留表示形式，它是支持 ADO. NET 的断开式、分布式数据方案的核心对象。

可以把 DataSet 对象看成数据库中部分数据及这些数据之间的关系在内存中的一个副本。无论数据库是何种类型，DataSet 都会提供一致的关系编程模型。可以在 DataSet 对象上进行读取操作，也可以进行插入、删除和修改等操作，并最终可将修改的内容反映到后台数据库中。DataSet 可以表示包括相关表、约束和表间关系在内的整个数据集，其对象模型如图 7-4 所示。

一个 DataSet 对象内可以包含 0 个或多个“表”。每个表为一个 DataTable 对象，这些表的集合由 Tables 属性表示。表与表之间的关系（主要是外键关系）也构成一个集合，

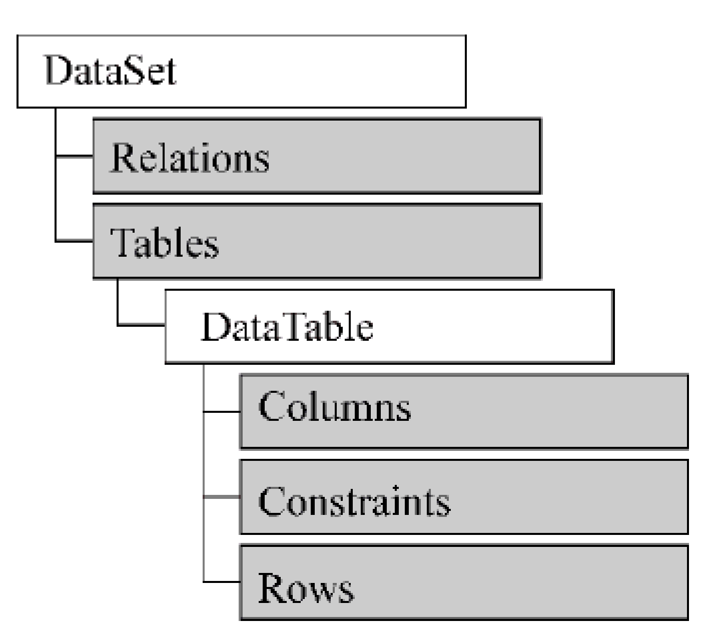


图 7-4 DataSet 对象模型(其中灰色部分为集合属性)

由 **Relations** 属性表示。

DataTable 在 System.Data 命名空间中定义，表示内存驻留数据的单个表，由列的集合(属性名为 Columns)以及约束的集合(属性名为 Constraints)来定义表的架构。

DataTable 还包含行的集合(属性名为 Rows)，表示表中的数据，其中每一行数据由一个 DataRow 对象表示。除了数据的当前状态之外，DataRow 还会保留其初始数据，以标识对行中存储的数据的更改，供最终修改数据库(调用 AcceptChanges 方法)时使用。

DataAdapter 对象是 ADO.NET 托管提供程序的组成部分。DataAdapter 对象用于在数据库和 DataSet 对象之间交换数据：将数据从数据库中读入 DataSet，然后将已更改的数据从 DataSet 写回数据库。DataAdapter 可以在任意数据库和 DataSet 之间移动数据。

通常，每个 DataAdapter 只在单个数据库表和 DataSet 内的单个 DataTable 对象之间交换数据(不使用多表联合查询)，如果 DataSet 包含多个 DataTable，可以用对应的多个 DataAdapter 向 DataSet 提供数据，并将其数据写回各个数据库表。这样做的好处是：可以使用 DataRelation 对象管理 DataSet 各表之间的关联关系(如级联更新等)，并可以在相关主记录和子记录之间移动。

DataAdapter 也需要通过打开的 Connection 对象才能读写数据库，因此同样需要为 DataAdapter 指明相关的 Connection 对象。但不需要显式地打开数据连接，当调用 DataAdapter 对象的 Fill 事件时，连接会自动打开。

使用 DataAdapter 对象可以读取、添加、修改和删除数据库中的数据。为使用户可以指定每种操作的执行方式，DataAdapter 对象支持 4 个属性：SelectCommand、InsertCommand、UpdateCommand 和 DeleteCommand。用户可以显式设置这些命令对象的文本，但这并不总是必要的。在很多情况下，如已经将 SelectCommand 属性指定为 select * from tablename，如果未指定 UpdateCommand、InsertCommand 或 DeleteCommand 对象，DataAdapter 可以在运行时自动生成适当的 SQL 语句。

使用 DataAdapter 主要包括读取和更新两种操作方式。

- 调用 DataAdapter 的 Fill 方法，将数据从数据库读到 DataSet。
- 调用 Update 方法，将对 DataSet 表的更改写回数据库。当调用该方法时，它将根据受影响的记录是新记录、已更改记录还是已删除记录来执行适当的 INSERT、UPDATE 或 DELETE 语句。

下面继续完善 7.3 节所创建的 UseADONET 网站。

第一步，在页面上增加一个按钮和一个 ListBox 控件，代码如下：

```
<br/>

<asp:Button ID="Button1" runat="server" OnClick="Button1_Click" Text="选课学生"/><br/>
```

```
<asp:ListBox ID="ListBox1" runat="server" Height="148px" Width="200px"></asp:ListBox>
```

第二步，为_Default 类再增加一个私有成员。

```
private static DataSet ds;
```

注意：该成员是静态成员，这样就可以在页面的各次调用之间保存数据。

将 FillData()函数改为：

```
private void FillData()
{
    //创建连接对象
    SqlConnection conn=new SqlConnection(connectionString);

    try
    {
        //打开连接
        conn.Open();

        //填充下拉式列表
        string cmdText="SELECT * FROM CLASS";
        SqlCommand command=new SqlCommand(cmdText, conn);
        SqlDataReader dr=command.ExecuteReader();
        while (dr.Read())
        {
            //向列表中添加 Item项
            DropDownList1.Items.Add(new ListItem(
                dr["CLASSID"]+"-"+dr["CLASSNAME"].ToString(),
                dr["CLASSID"].ToString()));
        }
        dr.Close();

        //填充 DataSet
        cmdText="SELECT * FROM student_class";
        SqlDataAdapter da=new SqlDataAdapter(cmdText, conn);
        ds=new DataSet("student_class");
        da.Fill(ds, "student_class");
    }
    catch (SqlException sqlex)
    {
        //显示错误信息
        Response.Write(sqlex.Message+"<br>");
    }
    finally
    {
        //关闭数据库链接
```

```
            conn.Close();
        }
    }
```

与前节代码相比，主要增加了填充 DataSet 的内容，但在代码格式上有了一定的变化。为了使代码能更清楚地表达功能，将 cmdText 等变量的说明和 SqlCommand、SqlDataAdapter 等对象的创建放到了 try 语句内部。

在功能比较单一的情况下，.NET 推荐仍然使用前节的编码风格。

第三步，为按钮的单击处理事件编写函数的代码如下：

```
protected void Button1_Click(object sender, EventArgs e)
{
    //创建 DataTable 对象
    DataTable dt=ds.Tables[0];
    //清除原有列表项
    ListBox1.Items.Clear();
    foreach (DataRow row in dt.Rows)
    {
        if (row["ClassID"].ToString() ==
DropDownList1.SelectedValue.ToString())
            //增加列表项
            ListBox1.Items.Add(new ListItem(
                row["UserID"]+"-"+row["ClassID"]+"-"
                +row["REGTIME"].ToString()));
    }
}
```

当每次单击按钮时，对 DataSet 中的第 1 个 DataTable(本例中也只有这一个)的各行(DataRow 对象)进行处理：如果该行的课程号与列表中所选的课程号相等，则将该行数据加入到 ListBox 中显示。

第四步，第一次执行网页，ListBox 显示为空。



图 7-5　选修某门课程的学生列表

在 DropDownList 中选择一门课程，单击“选课学生”按钮，页面重新加载，ListBox 中列出了选修当前课程的所有学生信息，如图 7-5 所示。

从上面例子也可以看出 DataReader 与 DataSet 的一个重要区别：DataReader 对象中的数据只能从前向后遍历一遍(只能使用一次)。如本例中，DataReader 对象中的数据只在页面第一次加载时用于填充 DropDownList 列表；DataSet 对象中的数据可以反复使用，如本例中，用户每次单击按钮，系统都会从 DataSet 对象中取出部分合适的数据加以显示。

另外，在本例中用程序控制遍历 DataTable 中的所有记录，判断哪些记录应该用于显示，这是为了演示 DataTable 的一般性操作方法。针对本例，更好

的办法是使用 DataView 对象。

DataView 对象与数据库中的视图相似，不同之处是：数据库视图经常是多个数据库表的联合视图；而 DataView 对象一般情况下只作用于一个 DataTable 对象，作为该 DataTable 对象行和列的子集的视图。除视图作用外，DataView 对象还提供排序、搜索和筛选等功能。

在上面示例中，可将按钮的单击事件处理函数修改为如下代码：

```
protected void Button1_Click(object sender, EventArgs e)
{
    //创建 DataView对象
    DataView dv= ds.Tables[0].DefaultView;
    //设置过滤表达式
    dv.RowFilter= "classid= '"+ DropDownList1.SelectedValue.ToString()+ "'";
    //清除原有列表项
    ListBox1.Items.Clear();
    foreach (DataRowView row in dv)
    {
        //增加列表项
        ListBox1.Items.Add(new ListItem(
            row["UserID"]+ "- "+ row["ClassID"]+ "- "
            + row["REGTIME"].ToString()));
    }
}
```

程序运行效果完全相同。

7.5　使用 Command 对象直接修改数据库

前两节介绍的方法主要应用于数据库的查询，中间也介绍了可以使用 DataSet 对象对数据库进行修改。其实，修改数据库最简单、最直接的方法是使用 Command 对象的 ExecuteNonQuery()方法。

将任意的 Update、Insert 和 Delete 命令以字符串的形式写入 Command 对象的 CommandText 属性，再调用 Command 对象的 ExecuteNonQuery()方法即可执行这些命令，并返回数据库中受影响的数据行数；若执行的语句不是 SQL 语句，则返回－1。请看下面实例。

创建一个名为 ExecuteNonQuery 的网站。网站的功能为：先显示 CLASS 表中的原始数据，再对 CLASS 表进行插入、修改和删除操作，并显示每步操作后的结果。

第一步，按前述方法创建 Web. config 文件，并增加连接字符串。

第二步，对 Default. aspx 的内容不做修改，直接打开其隐藏代码文件。

第三步，在 Default. aspx. cs 的开头加上对 System. Data. SqlClient 命名空间的引用。

将 Page_Load()函数的代码改为：

```
if (!Page.IsPostBack)
{
    TestExecuteNonQuery();
    Response.End();
}
```

实现 TestExecuteNonQuery()函数，代码如下：

```
/// <summary>
/// 测试用 ExecuteNonQuery 方法修改数据库
/// </summary>
private void TestExecuteNonQuery()
{
    string connectionString=ConfigurationManager.
ConnectionStrings["SQLConnectionString"].ConnectionString;
    //创建 SqlConnection
    SqlConnection conn=new SqlConnection(connectionString);

    try
    {
        //打开连接
        conn.Open();

        //显示初始数据
        Response.Write("课程表中的原始数据:<br>");
        DisplayData(conn);

        //插入数据
        string cmdText="INSERT INTO CLASS(CLASSID,CLASSNAME)VALUES('J11','一门新课')";
        SqlCommand command=new SqlCommand(cmdText, conn);
        int nCount=command.ExecuteNonQuery();
        Response.Write("<br>插入"+nCount.ToString()+"条新的数据之后:<br>");
        DisplayData(conn);

        //修改数据
        cmdText="UPDATE CLASS SET CLASSNAME='已经不是新课了' WHERE CLASSID='J11'";
        command.CommandText=cmdText;
        command.ExecuteNonQuery();
        Response.Write("<br>修改"+nCount.ToString()+"条数据之后:<br>");
        DisplayData(conn);

        //删除数据
        cmdText="DELETE FROM CLASS WHERE CLASSID='J11'";
        command.CommandText=cmdText;
        command.ExecuteNonQuery();
```

```
16:         Response.Write("<br>删除"+nCount.ToString()+"条数据之后:<br>");
17:         DisplayData(conn);
        }
        catch (SqlException sqlex)
        {
            //显示错误信息
            Response.Write(sqlex.Message+"<br>");
        }
        finally
        {
            //关闭数据库链接
            conn.Close();
        }
    }
```

其中:

01～02 行代码直接调用 DisplayData()函数显示课程表中的原始数据。

03～07 行代码先执行 INSERT 语句向数据库插入一条记录,再显示更新后的数据。

08～12 行代码先执行 UPDATE 语句对刚才新增的数据进行修改,再显示更新后的数据。

13～17 行代码先执行 DELETE 语句将刚才修改过的数据删除,再重新显示数据,此时数据已恢复至原始状态。

DisplayData()函数的实现代码如下:

```
/// <summary>
/// 显示数据库表中的数据
/// </summary>
/// <param name="conn">已打开的数据库连接</param>
private void DisplayData(SqlConnection conn)
{
    string cmdText="SELECT * FROM CLASS ORDER BY CLASSID";
    SqlCommand command=new SqlCommand(cmdText, conn);

    try
    {
        SqlDataReader dr=command.ExecuteReader();
        while (dr.Read())
        {
            Response.Write(dr["CLASSID"]+"-"+dr["CLASSNAME"]+"<br>");
        }
        dr.Close();
    }
    catch (SqlException sqlex)
    {
```

```
            //显示错误信息
            Response.Write(sqlex.Message+"<br>");
        }
    }
```

上述代码与7.3节中的示例代码功能相似，不再详细说明。

第四步，执行页面，显示如下信息：

```
课程表中的原始数据：
J01-C语言程序设计
J02-数据结构
J03-网络程序设计
J04-.NET程序设计

插入1条新的数据之后：
J01-C语言程序设计
J02-数据结构
J03-网络程序设计
J04-.NET程序设计
J11-一门新课

修改1条数据之后：
J01-C语言程序设计
J02-数据结构
J03-网络程序设计
J04-.NET程序设计
J11-已经不是新课了

删除1条数据之后：
J01-C语言程序设计
J02-数据结构
J03-网络程序设计
J04-.NET程序设计
```

习　　题

1. ADO.NET可以提供哪几种数据访问模式？各种数据访问模式分别有什么特点？
2. ADO.NET提供了哪些托管提供程序？
3. 请简述ADO.NET连接数据库的一般步骤。
4. 要想使用Oracle数据库的专用托管提供程序，在VS2005集成开发环境中需要进行哪些特殊设置？

5. 使用 VS2005 集成开发环境，自己动手创建 7.2.4 小节介绍的 ConnectToDatabase 网站，并观察网站的执行效果。

6. 参照 7.3 节的介绍，在习题 5 的基础上，使用 Command 对象和 DataReader 对象实现数据库查询功能。

7. 简述 SqlCommand 对象 CommandType 属性的作用及取值情况。

8. 简述 SqlCommand 对象 ExecuteNonQuery 方法的使用方法。

9. DataSet 对象有哪些功能？DataAdapter 对象与 DataSet 对象之间有何关联？

10. 写出向 DataSet 对象填充数据的典型代码。

11. 参照 7.4 节的介绍，在习题 6 的基础上，进一步使用 DataAdapter 对象与 DataSet 对象，实现如图 7-5 所示的执行效果。

12. 参照 7.5 节的介绍，使用 Command 对象实现该节所提到的 ExecuteNonQuery 网站，并观察网站的执行效果。

13. 写出使用 Command 对象执行 UPDATE 命令的典型代码。

第 8 章 chapter 8

Web 数据访问

第 7 章介绍了使用 ADO. NET 对象进行编程，直接对数据库中的数据进行操作的方法，这是 ASP. NET 数据库操作的基础。

其实，并不是所有数据库操作的细节都需要编程来控制。ASP. NET 提供了丰富的 Web 数据控件，这些控件屏蔽了大量的数据库操作细节，在不影响功能的同时较大地减少了代码量，从而使编程更加快捷和方便。

8.1 数据源控件

8.1.1 数据源控件概述

ASP. NET 中参与数据绑定的有两类服务器控件：数据源(DataSource)控件和数据绑定控件，这些控件完成 Web 数据访问的基础任务。

ASP. NET 包含一些 DataSource 控件，这些 DataSource 控件不呈现任何用户界面，而是充当不同类型数据源与网页上其他界面控件之间的中间方，这里的数据源是指数据库、XML 文件或中间层业务对象等。DataSource 控件对象可以用声明的方式(在网页文件中)或者以编程的方式(在代码隐藏文件中)定义。使用 DataSource 控件可以连接到数据源，无需编写代码即可实现以下功能：

- 从数据源中检索数据。
- 设置页面行为(如排序、分页、缓存等)。
- 更新、插入和删除数据。
- 使用运行时参数筛选数据。
- 允许其他界面控件绑定到 DataSource 控件，以便在网页中显示数据。

ASP. NET 包含支持不同数据绑定方案的 DataSource 控件，包括：

- ObjectDataSource：连接中间层对象或数据接口对象，使用 ObjectDataSource 可以创建依赖于中间层对象来管理数据的 Web 应用程序。
- SqlDataSource：连接 ADO. NET 托管数据提供程序，完成对 SQL Server、Oracle、OLE DB 或 ODBC 数据源的访问。
- AccessDataSource：连接 Access 数据库。

- XmlDataSource：连接 XML 数据源文件，一般为诸如 TreeView 或 Menu 等层次结构控件提供数据。
- SiteMapDataSource：与 ASP.NET 站点导航结合使用。

本书主要介绍 SqlDataSource 控件。

DataSource 控件不呈现任何用户界面，用户界面功能由数据绑定控件完成。数据绑定控件可以绑定到 DataSource 控件，并自动在页面请求生命周期的适当时机获取数据。数据绑定控件通过其 DataSourceID 属性连接到 DataSource 控件，然后即可利用 DataSource 控件所提供的功能，包括排序、分页、缓存、筛选、更新、删除和插入等。

本书前面所介绍的列表控件，如 BulletedList、CheckBoxList、DropDownList、ListBox 和 RadioButtonList 控件等，既可以作为一般控件使用，也可以作为数据绑定控件来使用。

ASP.NET 还提供一些专用的数据绑定控件，如 DataList、DetailsView、GridView 和 FormView 等，本章的重点是介绍前 3 种专用数据绑定控件，FormView 控件的使用将在 13.7 节介绍。

8.1.2 SqlDataSource 控件

SqlDataSource 控件使用 SQL 命令来检索和修改数据，可用于 SQL Server、Oracle、OLE DB 和 ODBC 等数据源。

SqlDataSource 控件可将检索结果作为 DataReader 或 DataSet 对象返回。当结果作为 DataSet 返回时，还可以对结果进行排序、筛选和缓存等操作。

以声明的方式在网页文件中创建 SqlDataSource 的方法，可用下面的实例来说明。

【例 8-1】 创建一个名为 UseGridView 的网站。

为网站创建一个新的页面 StudentManage1。最终要在此页面上完成对学生信息的管理。在本书的应用实例中，功能相同的页面也是 StudentManage1。在本节只介绍如何在此页面上创建 SqlDataSource 控件，管理功能将在下一节完成。

在工具箱的“数据”页中拖动一个 SqlDataSource 控件到页面上来，初始时代码为：

```
<asp:SqlDataSource ID="SqlDataSource1" runat="server">
</asp:SqlDataSource>
```

切换到设计视图，可以看到一个可视化的 SqlDataSource 控件。将鼠标移动到该控件右上角，可以看到一个小三角标记，该标记称为 SqlDataSource 控件的“智能标签”。单击智能标签，可以打开一个上下文相关的菜单，初始时如图 8-1 所示。



图 8-1 SqlDataSource 控件及其智能标签

单击“配置数据源”，弹出配置数据源向导，用户可在该向导的引导下，对数据源进行配置。

该向导的第一步是“选择你的数据连接”。

因为目前本网站还没有创建到数据库的连接，因此，需要单击“新建连接”按钮，进入“添加连接”对话框，添加一个新的数据库连接，如图 8-2 所示。

添加连接
输入信息以连接到选定的数据源，或单击"更改"选择另一个数据源和/或提供程序。
数据源(S):
Microsoft SQL Server (SqlClient)
更改(C)...
服务器名(E):
(local)
刷新(R)
登录到服务器
使用 Windows 身份验证(W)
使用 SQL Server 身份验证(Q)
用户名(U): sa
密码(P): ***
保存密码(S)
连接到一个数据库
选择或输入一个数据库名(D):
NetSchool
附加一个数据库文件(H):
浏览(B)...
逻辑名(L):
高级(V)...
测试连接(T)
确定
取消

图 8-2 “添加连接”对话框

参照图 8-2 进行数据库连接信息的配置：

- 数据源选择 Microsoft SQL Server(SqlClient)；
- 服务器名选择(local)；
- 登录方式选择“使用 SQL Server 身份验证”，用户名为 sa，输入正确的密码(本书假设为 abc)；
- 选择本书使用的示例数据库 NetSchool。

单击“测试连接”按钮进行测试，如果测试成功，则可单击“确定”按钮返回配置数据源向导，单击“下一步”继续。

向导提示“将连接字符串保存到应用程序配置文件中”，如图 8-3 所示。

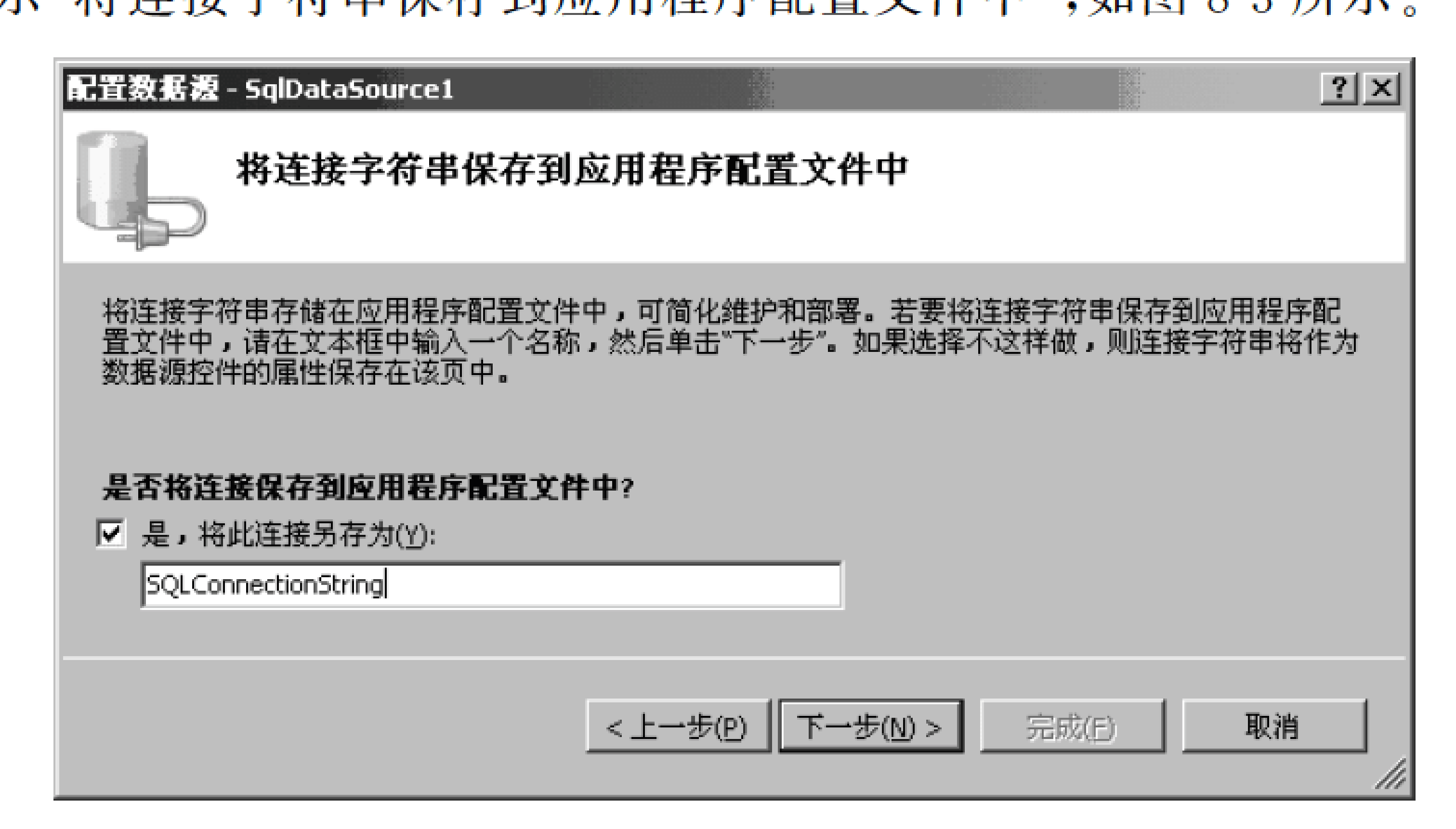


图 8-3 将连接字符串保存到应用程序配置文件中

选择“是”复选框，将连接字符串的名称改为 SQLConnectionString，单击“下一步”继续。

在“配置 Select 语句”步骤，界面如图 8-4 所示。

配置数据源 - SqlDataSource1

配置 Select 语句

希望如何从数据库中检索数据?

指定自定义 SQL 语句或存储过程(S)

指定来自表或视图的列(T)

名称(M):

STUDENT

列(O):

* REGTIME USERID SPECIALTY USERNAME REMARK PASSWORD SEX BIRTHDAY

只返回唯一行(E)

WHERE(W)...

ORDER BY(R)...

高级(V)...

SELECT 语句(L):

SELECT * FROM [STUDENT] ORDER BY [USERID]

< 上一步(P) 下一步(N) > 完成(F) 取消

图 8-4 配置 Select 语句

参照图 8-4，将数据源配置为选取 STUDENT 表的所有行和列。按 ORDER BY 按钮，为查询命令添加 ORDER BY 子句，如图 8-5 所示，可以看到将要生成的 SQL 语句为“SELECT * FROM [STUDENT] ORDER BY [USERID]”。在向导中单击“下一步”按钮继续。



图 8-5 “添加 ORDER BY 子句”对话框

在“测试查询”步骤，按“测试查询”按钮，如果示例数据库已经建立且前述步骤都配

置正确，应该能够显示结果数据集，如图 8-6 所示。

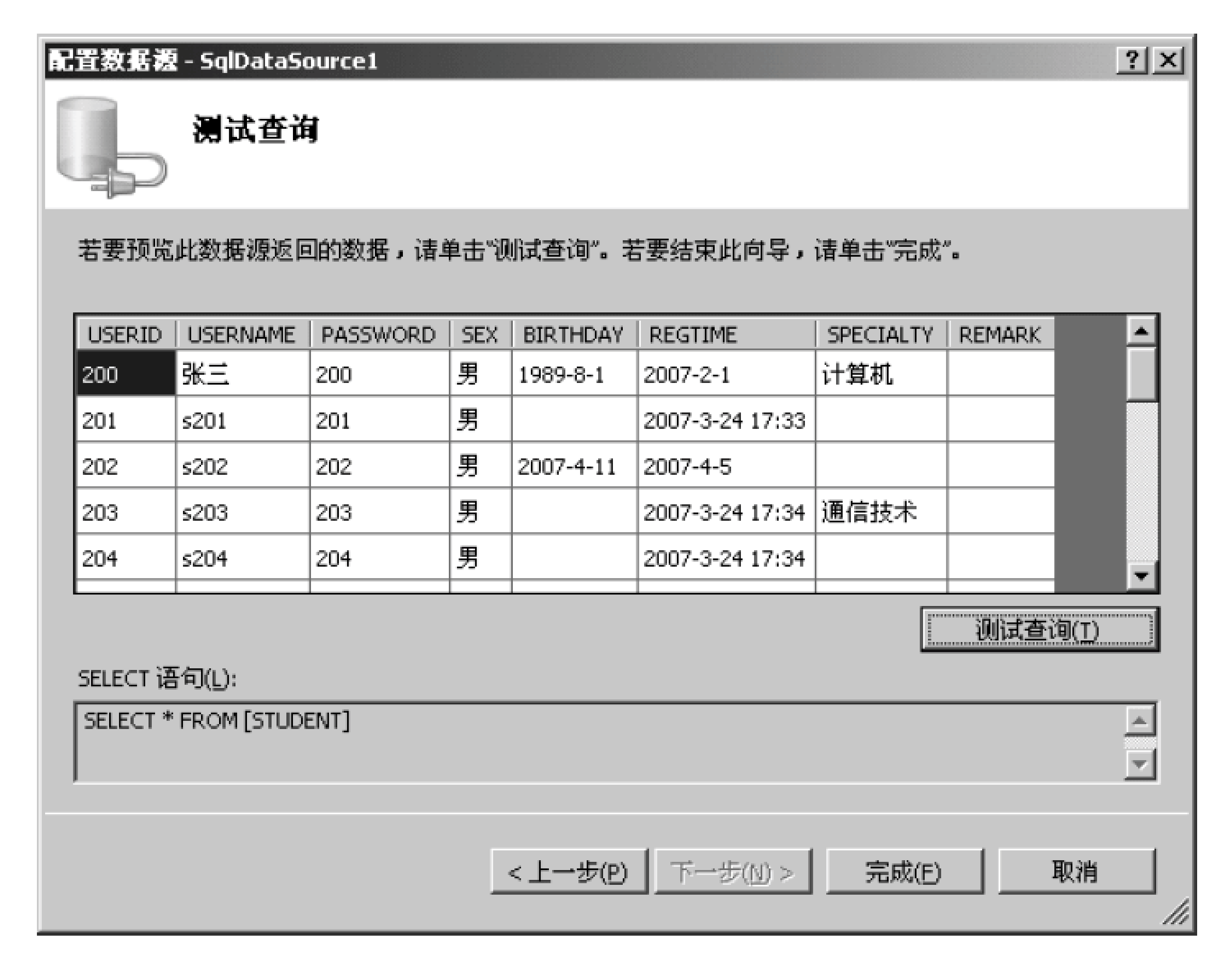


图 8-6 测试查询

单击“完成”按钮，完成对数据源的配置。

数据源配置完成后，可以看到网站项目中新增加了一个 Web. Config 文件。打开它，可以看到其中有已经配置好的数据库连接字符串 SQLConnectionString。

回到页面的源视图，可以看到 SqlDataSource 控件的代码已经被修改为：

```
<asp: SqlDataSource ID="SqlDataSource1" runat="server"
    ConnectionString="<%$ ConnectionStrings: SQLConnectionString %>"
    SelectCommand="SELECT * FROM [STUDENT] ORDER BY [USERID]">
</asp: SqlDataSource>
```

对 SqlDataSource 控件的配置先进行到这里，进一步的配置将在后续内容中逐步介绍。

8.2 GridView控件

8.2.1 常用属性和事件

GridView 控件是对旧版本中 DataGrid 控件的增强。与其他基本 Web 界面控件一样，GridView 控件同样在 System. Web. UI. WebControls 命名空间中定义。

使用 GridView 控件可以在“表”中显示数据源的值，其中每列表示一个字段，每行表示一条记录。GridView 控件还允许选择和编辑这些项以及对它们进行排序。

GridView 控件包括很多属性和事件，方便用户对其进行灵活的设计配置及运行程序控制。要全面介绍这些属性和事件需要太大的篇幅，本书只介绍其中最常用，特别是

在本书的应用实例中将要用到的部分属性和事件，如表 8-1 所示。更详细、全面的介绍请参考 MSDN。

表 8-1 GridView 控件常用的属性和事件

属性名称	说明
AllowPaging	是否启用分页功能，默认为 False
AllowSorting	是否启用排序功能，默认为 False
AutoGenerateColumns	是否为数据源中的每个字段自动创建绑定字段，默认为 True
AutoGenerateDeleteButton	每个数据行是否都带有“删除”按钮，默认为 False
AutoGenerateEditButton	每个数据行是否都带有“编辑”按钮，默认为 False
AutoGenerateSelectButton	每个数据行是否都带有“选择”按钮，默认为 False
BackColor、ForeColor、BorderColor、BorderStyle、BorderWidth 等	这些属性用于设置 GridView 控件的外观
Columns	DataControlField 对象的集合，表示 GridView 控件中的字段集
DataKeyNames	该属性值为一个数组，该数组包含了显示在 GridView 控件中记录的主键字段的名称
DataKeys	该值为一个 DataKey 对象，该 DataKey 对象表示 GridView 控件中的每一行数据的主键字段的值
DataSource	该属性值为一个 DataSource 对象，GridView 控件从该对象中获得数据
DataSourceID	GridView 控件要绑定到的 DataSource 控件的 ID，GridView 从该控件中获得数据
EmptyDataText	在 GridView 控件绑定到不包含任何记录的数据源时，所呈现在空数据行中的文本
PageSize	GridView 控件在每页上所显示记录的数目
SelectedIndex	GridView 控件中选中行的索引
SelectedRow	获取对 GridViewRow 对象的引用，该对象表示控件中的选中行
Visible	GridView 控件在页面上是否可见
事件名称	**说明**
DataBinding	当 GridView 绑定到数据源时发生
DataBound	在 GridView 绑定到数据源后发生
PageIndexChanged	在单击某个页导航按钮时(但在 GridView 控件处理分页操作之后)发生
RowDataBound	将 GridView 控件的行绑定到数据时发生

GridView 控件以一个表格的形式在浏览器上呈现数据。

默认情况下，GridView 控件一次性显示数据源中的所有数据。如果将 AllowPaging 属性值设置为 True，GridView 控件则可以分页显示数据，每页显示的记录数由 PageSize

属性指定。

在 GridView 控件所呈现的表格中会包含一个表头，一般情况下，表头中显示的是各字段的名称(可以设定为其他内容)。如果将 AllowSorting 属性设置为 True，表头中的字段名称将以超链的形式提供，单击某个字段的名称(超链)时，页面重新加载并按刚才所单击的字段进行排序。

默认情况下，GridView 控件的各列绑定到数据源的字段上显示数据。GridView 控件也可以包含一个命令字段，其中包含对当前记录数据进行操作的命令按钮，如“删除”、“编辑”和“选择”等。这些按钮是否显示，由 AutoGenerateDeleteButton、AutoGenerateEditButton 和 AutoGenerateSelectButton 属性确定。默认情况下，这些属性的值都为 False。

Columns 属性是一个集合属性，用来存储 GridView 控件中所有显式声明的字段。Columns 属性提供了以编程方式管理字段集合的方法。Columns 集合中各字段的顺序与 GridView 控件中各字段的显示顺序相同。

DataKeyNames 属性值为一个数组，该数组包含了数据源各主键字段的名称。当设置了 DataKeyNames 属性时，DataKeys 属性中则包含了在 DataKeyNames 属性中所指定的字段的值。

默认情况下，GridView 表格的各个单元中显示的是数据源中的数据；如果数据源中的相应数据为空，则不显示内容。如果设置了 EmptyDataText 属性，当检索到数据源中的空值时，将会在 GridView 控件的相应单元中显示 EmptyDataText 属性所设置的内容。

到目前为止，读者对专用数据绑定控件还不熟悉。要想在这种情况下充分理解上述属性和事件的确切含义有一定困难，下面结合实例进一步介绍。

8.2.2 GridView控件的基本应用

继续完成 8.2.1 节创建的 UseGridView 网站的 StudentManage1 页面。

从工具箱中拖一个 GridView 控件到页面上来，在源视图中可以看到 GridView 控件的初始代码为：

```
<asp:GridView ID="GridView1" runat="server">
</asp:GridView>
```

切换到设计视图，单出 GridView 控件的智能标签，在“选择数据源”中选择 SqlDataSource1，可以看到设计视图中的 GridView 控件外观发生了变化，主要是为 STUDENT 表的各字段生成了对应的绑定列。

再次打开 GridView 控件的智能标签，选择“启用分页”。

经过上述修改，GridView 控件的源代码改变为：

```
<asp:GridView ID="GridView1" runat="server" AutoGenerateColumns="False"
        DataKeyNames="USERID" DataSourceID="SqlDataSource1"
        AllowPaging="True">
```

```
<Columns>
    <asp: BoundField DataField= "USERID" HeaderText= "USERID"
        ReadOnly= "True" SortExpression= "USERID"/>
    <asp: BoundField DataField= "USERNAME" HeaderText= "USERNAME"
        SortExpression= "USERNAME"/>
    ……(略:其他各列)
</Columns>
</asp: GridView>
```

从上述代码中可以看出,由于已经显式地为各数据源字段创建了绑定字段(BoundField),因此将 AutoGenerateColumns 属性值置为 False,不再自动创建绑定字段。由于在设计视图中选择了 GridView 控件的“启用分页”选项,因此代码中包含了 AllowPaging="True"设置。

执行页面,运行效果如图 8-7 所示。

USERID	USERNAME	PASSWORD	SEX	BIRTHDAY	REGTIME	SPECIALTY	REMARK
200	张三	200	男	1989-8-1 0:00:00	2007-2-1 0:00:00	计算机	
201	s201	201	男		2007-3-24 17:33:28		
202	s202	202	男	2007-4-11 0:00:00	2007-4-5 0:00:00		
203	s203	203	男		2007-3-24 17:34:30	通信技术	
204	s204	204	男		2007-3-24 17:34:30		
205	s205	205	男		2007-3-24 17:34:30	数学	
206	s206	206	男		2007-3-24 17:34:30		
207	s207	207	男		2007-3-24 17:34:30		
208	s208	208	男		2007-3-24 17:34:30		
209	s209	209	男		2007-3-24 17:34:30		
1 2							

图 8-7 GridView 控件的执行效果(一)

从图 8-7 可以看出,GridView 控件的数据浏览功能已经基本具备了,但外观还不尽如人意。可以采用下面的步骤来使 GridView 控件的显示更加美观。

首先为页面增加一个标题,代码如下:

```
<h1 class= "title" style= "text- align: center">学生管理 1</h1>
```

在设计视图中改变 GridView 控件的属性:将 HorizontalAlign 改为“Center”,使 GridView 控件的显示居中;将 Width 属性改为“90%”;将 AllowSorting 属性改为“True”,允许单击各列的标题时针对该列进行排序;将 PageSize 属性改为“8”,使 GridView 的每页显示 8 条记录。

GridView 控件提供了众多属性用于外观控制,如 BackColor、ForeColor 和 BorderColor 等,对于具有美术天分的开发人员,完全可以通过手工配置这些属性来达到个性化的完美显示效果。但也许不必这么辛苦就能达到相当专业的显示效果。再次打开 GridView 控件的智能标签,选择“自动套用格式”,弹出如图 8-8 所示的对话框。

用户可以在左侧的“选择方案”列表中选择一种系统预置的外观方案(如“石板”),如果预览效果满意,按“确定”按钮回到设计视图,即可以看到外观的改善。

回到源视图,可以看到刚才在设计视图中修改属性对代码的影响。

手工修改各字段的 HeaderText 属性,将其改为中文。

再次执行页面,效果如图 8-9 所示。



图 8-8 “自动套用格式”对话框

学生管理1

编号	姓名	密码	性别	出生日期	注册时间	专业	备注
200	张三	200	男	1989-8-1 0:00:00	2007-2-1 0:00:00	计算机	
201	s201	201	男		2007-3-24 17:33:28		
202	s202	202	男	2007-4-11 0:00:00	2007-4-5 0:00:00		
203	s203	203	男		2007-3-24 17:34:30	通信技术	
204	s204	204	男		2007-3-24 17:34:30		
205	s205	205	男		2007-3-24 17:34:30	数学	
206	s206	206	男		2007-3-24 17:34:30		
207	s207	207	男		2007-3-24 17:34:30		
							1 2 3

图 8-9 GridView 控件的执行效果(二)

8.2.3 通过 GridView 控件修改数据

通过 8.2.2 小节的配置,GridView 控件的外观已经比较令人满意了。但要达到实用效果,还需要增强其数据控制能力,这就需要与 SqlDataSource 控件相配合。

在设计视图中打开 SqlDataSource1 的智能标签,选择“配置数据源”,可对数据源进行进一步的配置。

单击“下一步”按钮,到达“配置 Select 语句”步骤。单击“高级”按钮,在“高级 SQL 生成选项”对话框中选择“生成 INSERT、UPDATE 和 DELETE 语句”复选框。

完成对 SqlDataSource1 数据源的配置之后,在源视图中可以看到,其源代码已经改为:

```
<asp:SqlDataSource ID="SqlDataSource1" runat="server"
    ConnectionString="<%$ ConnectionStrings:SQLConnectionString %>"
    SelectCommand="SELECT * FROM [STUDENT] ORDER BY [USERID]"
    DeleteCommand="DELETE FROM [STUDENT] WHERE [USERID]=@USERID"
    InsertCommand="INSERT INTO [STUDENT] ([USERID], [USERNAME], [PASSWORD], [SEX], [BIRTHDAY],
[REGTIME], [SPECIALTY], [REMARK]) VALUES (@USERID, @USERNAME, @PASSWORD, @SEX, @BIRTHDAY, @REGTIME,
```

```
@SPECIALTY, @REMARK)"
    UpdateCommand="UPDATE [STUDENT] SET [USERNAME]=@USERNAME, [PASSWORD]=@PASSWORD, [SEX]=@SEX,
[BIRTHDAY]= @BIRTHDAY, [REGTIME] = @REGTIME, [SPECIALTY] = @SPECIALTY, [REMARK] = @REMARK WHERE
[USERID]=@USERID">
    <DeleteParameters>
        <asp:Parameter Name="USERID" Type="String"/>
    </DeleteParameters>
    <UpdateParameters>
        <asp:Parameter Name="USERNAME" Type="String"/>
        <asp:Parameter Name="PASSWORD" Type="String"/>
        …(略:其他参数)
    </UpdateParameters>
    <InsertParameters>
        <asp:Parameter Name="USERID" Type="String"/>
        <asp:Parameter Name="USERNAME" Type="String"/>
        …(略:其他参数)
    </InsertParameters>
</asp:SqlDataSource>
```

可以看出，主要是增加了三条命令：DeleteCommand、InsertCommand 和 UpdateCommand，并且给出了各命令的参数设置。

再次打开 GridView 控件的智能标签，选择“编辑列”，弹出“字段”对话框。

在“可用字段”列表中选择 CommandField，单击“添加”按钮，CommandField 出现在“选定的字段”列表中。

在右侧的属性列表中，将 ShowDeleteButton 属性、ShowEditButton 属性和 ShowInsertButton 属性都改为“True”，将 ButtonType 属性改为“Button”，将 SelectText 属性改为“所选课程”。

再次执行页面，效果如图 8-10 所示。

学生管理1

编号	姓名	密码	性别	出生日期	注册时间	专业	备注			
200	张三	200	男	1989-8-1 0:00:00	2007-2-1 0:00:00	计算机		编辑	删除	所选课程
201	s201	201	男		2007-3-24 17:33:28			编辑	删除	所选课程
202	s202	202	男	2007-4-11 0:00:00	2007-4-5 0:00:00			编辑	删除	所选课程
203	s203	203	男		2007-3-24 17:34:30	通信技术		编辑	删除	所选课程
204	s204	204	男		2007-3-24 17:34:30			编辑	删除	所选课程
205	s205	205	男		2007-3-24 17:34:30	数学		编辑	删除	所选课程
206	s206	206	男		2007-3-24 17:34:30			编辑	删除	所选课程
207	s207	207	男		2007-3-24 17:34:30			编辑	删除	所选课程
										1 2 3

图 8-10 GridView 控件的执行效果(三)

针对某一条记录，单击“编辑”按钮，可直接在当前行位置上对该记录数据进行编辑。单击“删除”按钮可删除当前记录。单击“所选课程”按钮，可以看到，页面重新加载后只是当前记录被选中了，没有其他实质性的功能执行。8.2.4 小节将继续完善“所选课程”功能。

8.2.4 多个 GridView和 SqlDataSource相互配合

ASP.NET 提供了多个 GridView 和多个 SqlDataSource 相互配合的能力。配合的方式有很多，可以达到丰富的功能效果，本小节只给出一个简单的实例来进行说明。

继续完善 8.2.3 小节的实例，希望达到如下功能：当按某个学生的“所选课程”按钮时，在页面的下部列出该学生所选的全部课程信息。

8.2.3 小节是先创建数据源，再创建 GridView 控件来使用该数据源。现在先创建 GridView 控件，再专为其创建新的数据源对象。

拖一个 GridView 控件到页面上原有内容的下部，可以看到其 ID 为 GridView2。

在设计视图上单击 GridView2 的智能标签，在“选择数据源”列表中选择“新建数据源”，进入“数据源配置向导”对话框，如图 8-11 所示。



图 8-11 “数据源配置向导”对话框

选择“数据库”，接受指定的数据源 ID，单击“确定”按钮继续。

在“选择你的数据连接”步骤选择 8.2.3 小节创建的 SQLConnectionString，单击“下一步”按钮继续。

在“配置 Select 语句”步骤选择“指定来自表或视图的列”，选择 STUDENT_CLASS 表的所有列。与 8.2.3 小节不同的是，需要为 Select 语句定义 WHERE 子句。

单击“WHERE”按钮，弹出“添加 WHERE 子句”对话框。在“列”处选择 USERID，在“源”处选择 Control。在右侧参数属性的“控件 ID”列表中选择 GridView1，单击“添加”按钮，可以看到 WHERE 子句列表中增加了一项内容。单击“确定”按钮结束配置。

完成数据源的配置之后，相关部分代码如下：

```
<asp: GridView ID= "GridView2" runat= "server" AutoGenerateColumns= "False" DataKeyNames= "USERID,
CLASSID" DataSourceID= "SqlDataSource2">
```

```
<Columns>
    <asp:BoundField DataField="USERID" HeaderText="USERID"
        ReadOnly="True" SortExpression="USERID"/>
    <asp:BoundField DataField="CLASSID" HeaderText="CLASSID"
        ReadOnly="True" SortExpression="CLASSID"/>
    <asp:BoundField DataField="REGTIME" HeaderText="REGTIME"
        SortExpression="REGTIME"/>
    <asp:BoundField DataField="GRADE" HeaderText="GRADE"
        SortExpression="GRADE"/>
</Columns>
</asp:GridView>
<asp:SqlDataSource ID="SqlDataSource2" runat="server"
    ConnectionString="<%$ ConnectionStrings:SQLConnectionString %>"
    SelectCommand="SELECT * FROM [STUDENT_CLASS] WHERE ([USERID]=@USERID)">
    <SelectParameters>
        <asp:ControlParameter ControlID="GridView1" Name="USERID"
            PropertyName="SelectedValue" Type="String"/>
    </SelectParameters>
</asp:SqlDataSource>
```

执行页面如图 8-12 所示，可以看到：当单击一个学生的“所选课程”按钮时，页面的下部就会列出他所选的课程列表。

学生管理1

编号	姓名	密码	性别	出生日期	注册时间	专业	备注			
200	张三	200	男	1989-8-1 0:00:00	2007-2-1 0:00:00	计算机		编辑	删除	所选课程
201	s201	201	男		2007-3-24 17:33:28			编辑	删除	所选课程
202	s202	202	男	2007-4-11 0:00:00	2007-4-5 0:00:00			编辑	删除	所选课程
203	s203	203	男		2007-3-24 17:34:30	通信技术		编辑	删除	所选课程
204	s204	204	男		2007-3-24 17:34:30			编辑	删除	所选课程
205	s205	205	男		2007-3-24 17:34:30	数学		编辑	删除	所选课程
206	s206	206	男		2007-3-24 17:34:30			编辑	删除	所选课程
207	s207	207	男		2007-3-24 17:34:30			编辑	删除	所选课程
										1 2 3

USERID	CLASSID	REGTIME	GRADE
201	J01	2007-3-24 19:31:00	78.60
201	J02	2007-3-24 19:32:54	99.20

图 8-12　GridView 控件的执行效果(四)

先忽略外观，来关注功能上的不足：在选课列表中只有课程的编号，而没有课程的名称。

如果想在列表中同时列出课程名称，则需要从两个数据库表中联合查询。

在设计视图上再次单击 GridView2 的智能标签，选择“配置数据源”。

在“配置 Select 语句”步骤选择“指定自定义 SQL 语句或存储过程”，单击“下一步”按钮继续。

在“定义自定义语句或存储过程”步骤，可利用“查询生成器”来生成 SQL 语句，也可以直接输入。

在“SELECT”页上单击“查询生成器”按钮，进入“查询生成器”对话框。可以看到，“查询生成器”分为上下几个部分，原来的SELECT语句已经存在。

在“查询生成器”最上面的部分中单击鼠标右键，在弹出式菜单中选择“添加表”，进入“添加表”对话框。在添加表对话框中选择CLASS表，按“添加”按钮，再单击“关闭”按钮，回到“查询生成器”对话框。可以看到“查询生成器”的上部增加了CLASS表的实体图。

在CLASS表的实体图中选中CLASSNAME字段，观察所生成SQL语句的变化。

单击“执行查询”按钮，在“值”栏中输入一个用户ID，如“202”，可显示查询结果如图8-13所示。

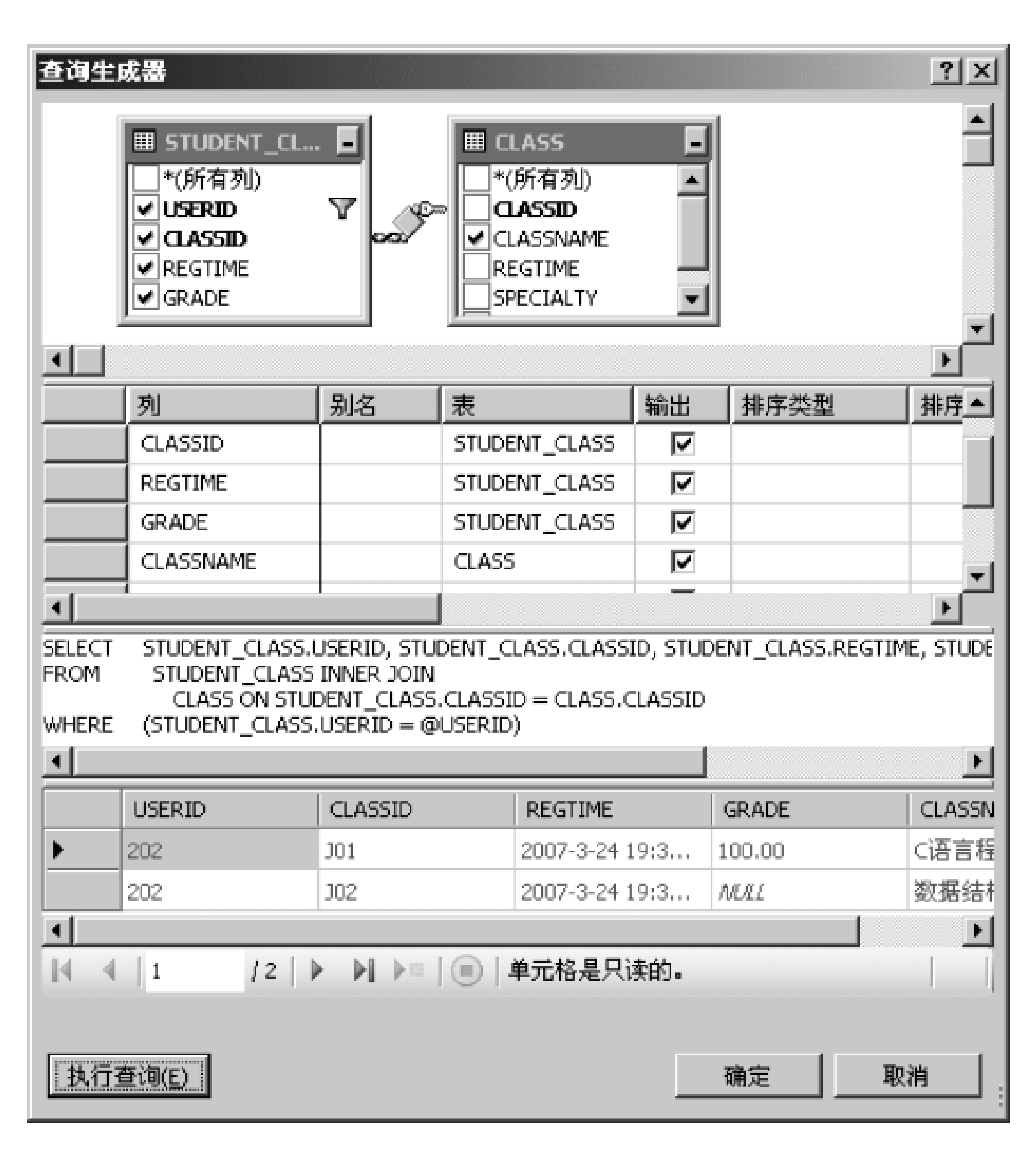


图8-13 查询生成器

单击“确定”按钮关闭“查询生成器”对话框。继续完成数据源的配置。完成后如果系统提问“是否刷新GridView2的列”，请选择“否”。

回到源视图，在GridView2控件中可以复制一个新的字段，也可以在原来CLASSID列的基础上修改，即将所有的CLASSID都改为CLASSNAME。再次执行页面，可以看到选课列表中列出的已经是课程名称了。

用与GridView1相似的方法修改GridView2的外观属性。修改后，相关代码如下：

```
<br/>
<asp:GridView ID="GridView2" runat="server" AutoGenerateColumns="False"
        DataKeyNames="USERID,CLASSID" DataSourceID="SqlDataSource2"
        HorizontalAlign="center" BackColor="White" BorderColor="#E7E7FF"
        BorderWidth="1px" CellPadding="3" GridLines="Horizontal"
        BorderStyle="None" Width="90%">
```

```
    <Columns>
        <asp: BoundField DataField="USERID" HeaderText="学号" ReadOnly="True"
            SortExpression="USERID"/>
        <asp: BoundField DataField="CLASSNAME" HeaderText="课程名称"
            ReadOnly="True" SortExpression="CLASSNAME"/>
        <asp: BoundField DataField="REGTIME" HeaderText="选课时间"
            SortExpression="REGTIME"/>
        <asp: BoundField DataField="GRADE" HeaderText="成绩"
            SortExpression="GRADE"/>
    </Columns>
    <FooterStyle BackColor="#B5C7DE" ForeColor="#4A3C8C"/>
    <SelectedRowStyle BackColor="#738A9C" ForeColor="#F7F7F7" Font-Bold="True"/>
    <PagerStyle BackColor="#E7E7FF" ForeColor="#4A3C8C"
        HorizontalAlign="Right"/>
    <HeaderStyle BackColor="#4A3C8C" Font-Bold="True" ForeColor="#F7F7F7"/>
    <AlternatingRowStyle BackColor="#F7F7F7"/>
    <RowStyle BackColor="#E7E7FF" ForeColor="#4A3C8C"/>
</asp: GridView>
<asp: SqlDataSource ID="SqlDataSource2" runat="server"
        ConnectionString="<%$ ConnectionStrings: SQLConnectionString %>"
    SelectCommand="SELECT STUDENT_CLASS.USERID, STUDENT_CLASS.CLASSID,
        STUDENT_CLASS.REGTIME, STUDENT_CLASS.GRADE, CLASS.CLASSNAME FROM
        STUDENT_CLASS INNER JOIN CLASS ON STUDENT_CLASS.CLASSID=CLASS.CLASSID
        WHERE (STUDENT_CLASS.USERID=@USERID)">
    <SelectParameters>
        <asp: ControlParameter ControlID="GridView1" Name="USERID"
            PropertyName="SelectedValue" Type="String"/>
    </SelectParameters>
</asp: SqlDataSource>
```

页面的执行效果如图 8-14 所示。

学生管理1

编号	姓名	密码	性别	出生日期	注册时间	专业	备注			
200	张三	200	男	1989-8-1 0:00:00	2007-2-1 0:00:00	计算机		编辑	删除	所选课程
201	s201	201	男		2007-3-24 17:33:28			编辑	删除	所选课程
202	s202	202	男	2007-4-11 0:00:00	2007-4-5 0:00:00			编辑	删除	所选课程
203	s203	203	男		2007-3-24 17:34:30	通信技术		编辑	删除	所选课程
204	s204	204	男		2007-3-24 17:34:30			编辑	删除	所选课程
205	s205	205	男		2007-3-24 17:34:30	数学		编辑	删除	所选课程
206	s206	206	男		2007-3-24 17:34:30			编辑	删除	所选课程
207	s207	207	男		2007-3-24 17:34:30			编辑	删除	所选课程

1 2 3

学号	课程名称	选课时间	成绩
202	C语言程序设计	2007-3-24 19:31:55	100.00
202	数据结构	2007-3-24 19:32:54	

图 8-14 GridView 控件的执行效果(五)

8.2.5 对 GridView 控件编程

到 8.2.4 小节为止,已经使用 GridView 控件达到了很强的功能效果,但还没有为它编写一行程序。这并不是说 GridView 控件的编程控制能力差,实际上,可以针对 GridView 进行编程,来实现更加丰富的功能。

在 8.2.3 小节的基础上继续修改示例程序。

为 UseGridView 网站创建一个新的页面 StudentManage2,目的是用另一种方法完成对学生信息的管理。在本书的应用实例中,StudentManage2 页面也可以完成同样的功能。

将 StudentManage1 页面中的<h1>部分、SqlDataSource1 部分和 GridView1 部分的代码复制到 StudentManage2 中,StudentManage2 页面现在就能够执行。

将<h1>部分的标题改为"学生管理 2"。

将 GridView1 的 CommandField 列部分改为只剩下一个按钮,代码为:

```
<asp:CommandField ButtonType="Button" ShowSelectButton="True"
    SelectText="修改"/>
```

为<div>加上显示居中属性:

```
<div style="text-align: center">
```

在 GridView1 的下面增加一个按钮控件,并指定其单击处理事件:

```
<asp:Button ID="btnAdd" runat="server" Text="新增"
    OnClick="btnAdd_Click"/>
```

在"新增"按钮下面增加一个 Panel 控件:

```
<asp:Panel ID="Panel1" runat="server" style="text-align: center"
    Width="90%">
</asp:Panel>
```

在 Panel 控件上增加一个 table,table 中包括对学生信息的编辑界面,其代码如下:

```
<table width="100%" border="1" style="color: #4A3C8C; background-color: #E7E7FF;">
    <tr>
        <td style="width: 10%">
            <span style="color: navy">用户号:</span>
        </td>
        <td style="width: 23%">
            <asp:TextBox ID="USERIDTextBox" runat="server" Width="97%" Text=''>
            </asp:TextBox>
        </td>
        <td style="width: 10%">
            <span style="color: navy">姓名:</span>
        </td>
```

```
<td style="width: 23% ">
    <asp: TextBox ID="USERNAMETextBox" runat="server" Width="97% " Text=''>
    </asp: TextBox>
</td>
<td style="width: 12% ">
    <span style="color: navy">密码：</span>
</td>
<td style="width: 22% ">
    <asp: TextBox ID="PASSWORDTextBox" runat="server" Width="97% " Text=''>
    </asp: TextBox>
</td>
</tr>
<tr>
<td>
    <span style="color: navy">性别：</span>
</td>
<td>
    <asp: TextBox ID="SEXTextBox" runat="server" Width="97% " Text=''>
    </asp: TextBox>
</td>
<td>
    <span style="color: navy">生日：</span>
</td>
<td>
    <asp: TextBox ID="BIRTHDAYTextBox" runat="server" Width="97% " Text=''>
    </asp: TextBox>
</td>
<td>
    <span style="color: navy">注册时间：</span>
</td>
<td>
    <asp: TextBox ID="REGTIMETextBox" runat="server" Width="97% " Text=''>
    </asp: TextBox><br/>
</td>
</tr>
<tr>
<td>
    <span style="color: navy">专业：</span>
</td>
<td>
    <asp: TextBox ID="SPECIALTYTextBox" runat="server" Width="97% "
        Text=''>
    </asp: TextBox><br/>
</td>
```

```
                <td>
                    <span style="color: navy">备注:</span>
                </td>
                <td colspan="3">
                    <asp:TextBox ID="REMARKTextBox" runat="server" Width="98%" Text=''
                        Rows="3" TextMode="MultiLine">
                    </asp:TextBox><br/>
                </td>
            </tr>
            <tr>
                <td colspan="6">
                    <asp:Button ID="btnSave" runat="server"
                        OnClick="btnSave_Click" Text="保存"/>
                    <asp:Button ID="btnCancel" runat="server" Text="取消"
                        OnClick="btnCancel_Click"/>
                    <asp:Button ID="btnDelete" runat="server" Text="删除"
                        OnClick="btnDelete_Click"/>
                </td>
            </tr>
        </table>
```

下面针对不同的功能要求,为该页面编写程序。

首先,如果用户没有要求修改或新增学生信息,页面下部的 Panel1 部分应该是不显示的,如页面第一次加载的时候和修改的信息提交之后。先在页面的 Page_Load 函数中控制 Panel1 不可见,其他情况下 Panel1 是否可见在各按钮的单击事件处理函数中控制。

为页面的 Page_Load 函数增加如下代码:

```
if (!Page.IsPostBack)
{
    this.Panel1.Visible=False;
}
```

可能已经注意到,在显示学生信息时,学生的生日字段在显示时也会默认地显示其时间部分,这显然是不合适的。要控制生日字段的显示,可以为 GridView1 的 RowDataBound 事件增加处理函数。

以前要为某个控件增加处理函数时,总是在设计视图中双击该控件,系统会自动地转到代码隐藏文件中,为该控件的默认事件增加处理函数,例如,为按钮的单击事件增加处理函数。但编程并不总是处理控件的默认事件,在此介绍另一种进入控件事件编程的方法(4.2 节已经简单介绍过):在设计视图上选中 GridView1 控件,单击"属性"窗口上部的闪电图标,窗口中列出 GridView1 控件的所有事件。双击 RowDataBound 行,系统会自动转到代码隐藏文件中,并为 GridView1 的 RowDataBound 事件增加处理函数,默认的函数名为 GridView1_RowDataBound。

为 GridView1_RowDataBound 函数增加代码如下:

```
//如果是数据行则进行处理
if (e.Row.RowType==DataControlRowType.DataRow)
{
    //取左数第 4个 (从 0开始计数)单元格 (生日)的数据
    TableCell cell=e.Row.Cells[4];
    string s=cell.Text;
    //删除空格以后的部分
    int i=s.IndexOf(' ');
    int j=s.Length;
    if (i>=0)
        s=s.Remove(i, j-i);
    //将修改后的文本写回单元格
    cell.Text=s;
}
```

执行页面,界面如图 8-15 所示。

学生管理2

编号	姓名	密码	性别	出生日期	注册时间	专业	备注	
200	张三	200	男	1989-8-1	2007-2-1 0:00:00	计算机		修改
201	s201	201	男		2007-3-24 17:33:28			修改
202	s202	202	男	2007-4-11	2007-4-5 0:00:00			修改
203	s203	203	男		2007-3-24 17:34:30	通信技术		修改
204	s204	204	男		2007-3-24 17:34:30			修改
205	s205	205	男		2007-3-24 17:34:30	数学		修改
206	s206	206	男		2007-3-24 17:34:30			修改
207	s207	207	男		2007-3-24 17:34:30			修改

1 2 3

新增

图 8-15　GridView 控件的执行效果(六)

当用户单击“修改”按钮时,页面的下部出现当前记录的信息修改界面,以当前记录的原始值为初值。包括 3 个功能按钮,单击“保存”按钮将修改写到数据库中;单击“取消”按钮使界面的信息修改部分不可见;单击“删除”按钮删除当前记录。

当用户单击“修改”按钮时,系统的处理是:当前记录被选中,并触发 GridView1 的 SelectedIndexChanged 事件。为 GridView1 的 SelectedIndexChanged 事件增加处理函数 GridView1_RowDataBound,其代码为:

```
//置 Panel1为可见
this.Panel1.Visible=True;
//置“删除”按钮可见。因为也许在“新增”时已经将其置为不可见
this.btnDelete.Visible=True;

//从 GridView1的当前行取原值
GridViewRow row=GridView1.SelectedRow;
```

```
String USERID= row.Cells[1].Text;
String USERNAME= row.Cells[2].Text;
String PASSWORD= row.Cells[3].Text;
//对可能为空的字段需要进行特殊处理,如下:
String SEX= row.Cells[4].Text;
if (SEX == " ") SEX= "";
String BIRTHDAY= row.Cells[5].Text;
if (BIRTHDAY == " ") BIRTHDAY= "";
String REGTIME= row.Cells[6].Text;
if (REGTIME == " ") REGTIME= "";
String SPECIALTY= row.Cells[7].Text;
if (SPECIALTY == " ") SPECIALTY= "";
String REMARK= row.Cells[8].Text;
if (REMARK == " ") REMARK= "";

//为修改部分置初值
//USERID 为主键字段,不允许修改
this.USERIDTextBox.ReadOnly= True;
this.USERIDTextBox.Text= USERID;
this.USERNAMETextBox.Text= USERNAME;
this.PASSWORDTextBox.Text= PASSWORD;
this.SEXTextBox.Text= SEX;
this.BIRTHDAYTextBox.Text= BIRTHDAY;
this.REGTIMETextBox.Text= REGTIME;
this.SPECIALTYTextBox.Text= SPECIALTY;
this.REMARKTextBox.Text= REMARK;
```

当用户单击“新增”按钮时,页面的下部出现包含一条空记录的新增界面。界面包括“保存”和“取消”两个功能按钮,这就意味着需要将“删除”按钮隐藏。

为“新增”按钮的单击事件处理函数编写代码如下:

```
//置 Panel1 为可见
this.Panel1.Visible= True;
//置“删除”按钮不可见
this.btnDelete.Visible= False;

//将各输入域置空。因为也许以前在编辑时已经为各域赋了值
this.USERIDTextBox.ReadOnly= False;
this.USERIDTextBox.Text= "";
this.USERNAMETextBox.Text= "";
this.PASSWORDTextBox.Text= "";
this.SEXTextBox.Text= "";
this.BIRTHDAYTextBox.Text= "";
this.REGTIMETextBox.Text= "";
this.SPECIALTYTextBox.Text= "";
```

```
this.REMARKTextBox.Text="";
```

当用户单击“保存”按钮时，需要将用户输入的信息写到数据库中。这需要对当前的编辑状态是“修改”还是“新增”进行判断，进而做出不同的处理。

第 7 章介绍了通过 ADO. NET 对象来直接修改数据库的方法，该方法在此处完全适用。但此处希望介绍另一种方法：使用 DataSource 控件本身所提供的功能来修改数据库。

为“保存”按钮的单击事件处理函数编写代码如下：

```
//取得修改后的值
String USERID=Request.Params["USERIDTextBox"].ToString();
String USERNAME=Request.Params["USERNAMETextBox"].ToString();
String PASSWORD=Request.Params["PASSWORDTextBox"].ToString();
String SEX=Request.Params["SEXTextBox"].ToString();
String BIRTHDAY=Request.Params["BIRTHDAYTextBox"].ToString();
String REGTIME=Request.Params["REGTIMETextBox"].ToString();
String SPECIALTY=Request.Params["SPECIALTYTextBox"].ToString();
String REMARK=Request.Params["REMARKTextBox"].ToString();

//根据 USERID 域是否为只读来判断编辑状态
//如果不只读(说明是“新增”状态)
if (!this.USERIDTextBox.ReadOnly)
{
    //置插入命令的参数
    System.Web.UI.WebControls.Parameter param =
    SqlDataSource1.InsertParameters["USERID"];
    param.DefaultValue=USERID;
    param=SqlDataSource1.InsertParameters["USERNAME"];
    param.DefaultValue=USERNAME;
    param=SqlDataSource1.InsertParameters["PASSWORD"];
    param.DefaultValue=PASSWORD;
    param=SqlDataSource1.InsertParameters["SEX"];
    param.DefaultValue=SEX;
    param=SqlDataSource1.InsertParameters["BIRTHDAY"];
    param.DefaultValue=BIRTHDAY;
    param=SqlDataSource1.InsertParameters["REGTIME"];
    param.DefaultValue=REGTIME;
    param=SqlDataSource1.InsertParameters["SPECIALTY"];
    param.DefaultValue=SPECIALTY;
    param=SqlDataSource1.InsertParameters["REMARK"];
    param.DefaultValue=REMARK;
    //调用 SqlDataSource 的插入事件,向数据库插入记录
    SqlDataSource1.Insert();
}
```

```
//如果只读(说明是"修改"状态)
if (this.USERIDTextBox.ReadOnly)
{
    //置修改命令的参数
    System.Web.UI.WebControls.Parameter param =
        SqlDataSource1.UpdateParameters["USERID"];
    param.DefaultValue= USERID;
    param= SqlDataSource1.UpdateParameters["USERNAME"];
    param.DefaultValue= USERNAME;
    param= SqlDataSource1.UpdateParameters["PASSWORD"];
    param.DefaultValue= PASSWORD;
    param= SqlDataSource1.UpdateParameters["SEX"];
    param.DefaultValue= SEX;
    param= SqlDataSource1.UpdateParameters["BIRTHDAY"];
    param.DefaultValue= BIRTHDAY;
    param= SqlDataSource1.UpdateParameters["REGTIME"];
    param.DefaultValue= REGTIME;
    param= SqlDataSource1.UpdateParameters["SPECIALTY"];
    param.DefaultValue= SPECIALTY;
    param= SqlDataSource1.UpdateParameters["REMARK"];
    param.DefaultValue= REMARK;
    //调用 SqlDataSource 的修改事件,修改数据库中的记录
    SqlDataSource1.Update();
}

//修改完成,将 Panel1 置为不可见
this.Panel1.Visible= False;
```

当用户单击"删除"按钮时,同样采用 DataSource 控件本身所提供的功能修改数据库。为"删除"按钮的单击事件处理函数编写代码如下:

```
//置删除命令的参数
String USERID= Request.Params["USERIDTextBox"].ToString();
System.Web.UI.WebControls.Parameter param =
    SqlDataSource1.DeleteParameters["USERID"];
param.DefaultValue= USERID;
//调用 SqlDataSource 的删除事件,删除数据库中的记录
SqlDataSource1.Delete();

//删除完成,将 Panel1 置为不可见
this.Panel1.Visible= False;
```

当用户单击“取消”按钮时，只需要将界面的编辑部分隐藏即可。为“取消”按钮的单击事件处理函数编写代码如下：

```
//取消操作,将 Panel1 置为不可见
this.Panel1.Visible= False;
```

完成上述工作后，当进行记录修改时，界面如图 8-16 所示。

学生管理2

编号	姓名	密码	性别	出生日期	注册时间	专业	备注	
200	张三	200	男	1989-8-1	2007-2-1 0:00:00	计算机		修改
201	s201	201	男		2007-3-24 17:33:28			修改
202	s202	202	男	2007-4-11	2007-4-5 0:00:00			修改
203	s203	203	男		2007-3-24 17:34:30	通信技术		修改
204	s204	204	男		2007-3-24 17:34:30			修改
205	s205	205	男		2007-3-24 17:34:30	数学		修改
206	s206	206	男		2007-3-24 17:34:30			修改
207	s207	207	男		2007-3-24 17:34:30			修改

1 2 3

新增

用户号:	200	姓名:	张三	密码:	200
性别:	男	生日:	1989-8-1	注册时间:	2007-2-1 0:00:00
专业:	计算机	备注:			

保存 取消 删除

图 8-16　GridView 控件的执行效果(七)

本小节实现的学生管理功能看起来并不是很实用，但这并不是说所使用的实现方法不实用，这与应用场合有关。本节是介绍技术，同样的技术应用在其他场合，如课程管理就是很实用的功能了，详见 13.9 节。这再一次提醒大家，掌握了基本的开发技术之后，针对具体的需求，选择合适的技术是很重要的。

从上面的两个例子可以看出，当需要列表、分页、排序显示大量数据时，使用 GridView 控件是一个很好的选择。

GridView 控件本身提供了很强的功能，还可以通过编程的方式完成对数据更复杂的操作。

在掌握了 GridView 控件的基本使用与编程技巧之后，再与其他编程技巧相结合，就可以达到更加完善、实用的效果。如本书应用实例的 StudentManage3 页面就是一种使用 GridView 控件，对大量数据进行更加有效管理的示例，将在本书的 13.8 节介绍。

由于控件本身提供了内置的分页、排序等功能，因此，GridView 控件适合于管理记录较多的数据库表。由于 GridView 控件封装良好，从另一个方面来看，不容易在字段上增加各种灵活操作，因此适合于对没有更多关联关系的数据库表的管理。在需要对记录字段进行灵活操作的场合，可以选择使用下一节将要介绍的 DataList 控件或第 9 章将要介绍的 Repeater 控件。

8.3 DataList 控件

DataList 控件与 GirdView 控件在功能上比较相似，应用场合也相近。DataList 控件在界面和操作上更加灵活，可以达到更加个性化的效果。从另一个方面来看，使用 DataList 控件需要手工控制的地方更多，也需要编写更多的程序代码。

8.3.1 DataList 控件的模板和事件

DataList 控件可用于模板化地列表显示数据，可用自定义的格式显示数据库的各行信息。使用 DataList 控件，可通过所创建的模板来定义数据显示布局。所谓模板，就是用来控制显示数据库中每条记录的 HTML。

使用 DataList 控件时，将每条数据库记录作为一个“项”来处理。DataList 控件支持几种类型的项，包括：(一般)项、交替项、选定项和编辑项等，可以为各种类型的项创建不同的模板。

也可以使用标题、脚注和分隔符模板自定义 DataList 的整体外观。通过在模板中使用 Button 控件，可将列表项连接到代码，而这些代码允许用户在显示、选择和编辑模式之间进行切换，还可以自定义该控件以支持其他功能。

DataList 控件可用于列表操作任何重复结构中的数据，它使用 HTML 的 table 元素在列表中显示数据记录。但是，若要更精确地控制用于显示的 HTML，可以选择使用 Repeater 控件，Repeater 控件将在下一章介绍。

表 8-2 列出了 DataList 控件所支持的模板。

表 8-2 DataList 控件支持的模板

模板名称	说明
ItemTemplate	包含一些 HTML 元素和控件，必须定义，是每一项的内容和布局的默认定义
AlternatingItemTemplate	包含一些 HTML 元素和控件，为交替行创建不同的外观，例如指定一个与 ItemTemplate 中指定的颜色不同的背景色
SelectedItemTemplate	定义当前选中行的外观。通常，用户可以使用此模板来通过不同的背景色或字体颜色直观地区分选定的行，还可以通过显示数据源中的其他字段来展开该项
EditItemTemplate	指定当某项处于编辑模式时的布局。此模板通常包含一些编辑控件，如 TextBox 等
HeaderTemplate	定义列表开始处呈现的文本和控件
FooterTemplate	定义列表结束处呈现的文本和控件
SeparatorTemplate	定义在每项之间呈现的元素。典型的示例可能是一条直线(使用 HR 元素)

每个模板都支持其自己的样式对象，样式对象包含一系列外观属性的定义，可以在设计时和运行时进行设置。与表 8-2 相对应，样式对象包括：

- ItemStyle；
- AlternatingItemStyle；
- SelectedItemStyle；
- EditItemStyle；
- HeaderStyle；
- FooterStyle；
- SeparatorStyle。

DataList 控件支持的公共事件如表 8-3 所示。

表 8-3 DataList 控件的常用公共事件

事件名称	说明
CancelCommand	当单击 DataList 控件中编辑项的 Cancel 按钮(取消编辑)时发生
DataBinding	当 DataList 控件绑定到数据源时发生
DeleteCommand	当单击 DataList 控件中某一项的 Delete 按钮(删除该项)时发生
EditCommand	当单击 DataList 控件中某一项的 Edit 按钮(使该项进入编辑状态)时发生
ItemCommand	当单击 DataList 控件中的任一按钮时发生
ItemCreated	当在 DataList 控件中创建项时在服务器上发生
ItemDataBound	当数据绑定到 DataList 控件的项时发生
SelectedIndexChanged	在两次服务器发送之间,如果 DataList 控件中选择了不同的项,则触发该事件
UpdateCommand	当单击 DataList 控件中编辑项的 Update 按钮(将修改提交到数据库)时发生

其中有些事件是在 DataList 控件生命周期的各步骤上由系统触发的，如 ItemCreated、ItemDataBound 等；而有些事件是为了响应用户的命令，包括 EditCommand、DeleteCommand、UpdateCommand 和 CancelCommand。若要引发这些事件,可将 Button、LinkButton 或 ImageButton 等控件添加到 DataList 控件的模板中,并将这些按钮的 CommandName 属性设置为某个关键字,如 edit、delete、update 或 cancel。当用户单击某个按钮时,就会引发 CommandName 属性所对应的事件。

DataList 控件还支持 ItemCommand 事件,当用户单击某个没有预定义命令(如 edit 或 delete)的按钮时将触发该事件,处理方法是：将该按钮的 CommandName 属性设置为一个自定义的值,然后在 ItemCommand 事件处理程序中根据这个值作出相应的处理。

8.3.2 DataList 控件的基本应用

【例 8-2】 创建一个名为 UseDataList 的网站。

为网站创建一个新的页面 ManagerManage1。最终要在此页面上完成对系统管理员信息的管理。在本应用实例中,功能相同的页面也是 ManagerManage1。

从工具箱拖一个 DataList 控件到页面上来,初始代码如下：

```
<asp: DataList ID= "DataList1" runat= "server">
```

```
</asp: DataList>
```

切换到设计视图，打开 DataList1 的智能标签，按照 8.3.1 小节的方法，新建一个称为 SqlDataSource1 的数据源：连接字符串的名称用 SQLConnectionString，指定选取 MANAGER 表的所有字段，按 USERID 字段排序。

为 DataList1 创建了数据源后，在设计视图中可以看到其预览形式已经发生了变化——为记录创建了标签。回到源视图可以看到，除了增加了一个数据源对象外，DataList1 的代码也有较大的变化。代码如下：

```
<asp: DataList ID="DataList1" runat="server" DataKeyField="USERID"
        DataSourceID="SqlDataSource1">
    <ItemTemplate>
        USERID:
        <asp: Label ID="USERIDLabel" runat="server" Text='<%#
    Eval("USERID") %>'></asp: Label><br/>
        USERNAME:
        <asp: Label ID="USERNAMELabel" runat="server" Text='<%#
    Eval("USERNAME") %>'></asp: Label><br/>
        PASSWORD:
        <asp: Label ID="PASSWORDLabel" runat="server" Text='<%#
    Eval("PASSWORD") %>'></asp: Label><br/>
        SEX:
        <asp: Label ID="SEXLabel" runat="server" Text='<%#
    Eval("SEX") %>'></asp: Label><br/>
        BIRTHDAY:
        <asp: Label ID="BIRTHDAYLabel" runat="server" Text='<%#
    Eval("BIRTHDAY") %>'></asp: Label><br/>
        DUTY:
        <asp: Label ID="DUTYLabel" runat="server" Text='<%#
    Eval("DUTY") %>'></asp: Label><br/>
        SPECIALTY:
            <asp: Label ID="SPECIALTYLabel" runat =
"server" Text='<%# Eval ("SPECIALTY") % > ' >
</asp: Label><br/>
        REMARK:
            <asp: Label ID="REMARKLabel" runat =
"server" Text='<%# Eval("REMARK") %>'></asp: Label><
br/>
        <br/>
    </ItemTemplate>
</asp: DataList>
```

主要是增加了对数据源的绑定和生成了 ItemTemplate 模板。执行页面，界面如图 8-17 所示。



图 8-17 DataList 控件的执行效果(一)

该页面已经能够从数据库中取数据并加以显示,但显示效果不尽如人意。下面就开始着手改善 DataList 的外观。

为页面增加一个标题,代码如下:

```
<h1 align="center">管理人员列表 1</h1>
```

将 DataList1 的 ItemTemplate 模板的内容手工改为在一个 table 中显示,代码如下:

```
<ItemTemplate>
    <table border="1" width="100%">
        <tr>
    <td Width="10%" align=center valign=middle>
        <span style="color: navy">用户号</span></td>
    <td Width="23%"><asp: Label ID="USERIDLabel" runat="server"
        Text='<%# Eval("USERID") %>'></asp: Label></td>
    <td Width="10%" align=center valign=middle>
        <span style="color: navy">姓名</span></td>
    <td Width="23%"><asp: Label ID="USERNAMELabel" runat="server"
        Text='<%# Eval("USERNAME") %>'></asp: Label></td>
    <td Width="10%" align=center valign=middle>
        <span style="color: navy">密码</span></td>
    <td Width="24%"><asp: Label ID="PASSWORDLabel" runat="server"
        Text='<%# Eval("PASSWORD") %>'></asp: Label></td>
</tr>
<tr>
    <td align=center valign=middle>
        <span style="color: navy">性别</span></td>
    <td><asp: Label ID="SEXLabel" runat="server"
        Text='<%# Eval("SEX") %>'></asp: Label>  </td>
    <td align=center valign=middle>
        <span style="color: navy">生日</span></td>
    <td><asp: Label ID="BIRTHDAYLabel" runat="server"
        Text='<%# Eval("BIRTHDAY") %>'></asp: Label>  </td>
    <td align=center valign=middle>
        <span style="color: navy">职务</span></td>
    <td><asp: Label ID="DUTYLabel" runat="server"
        Text='<%# Eval("DUTY") %>'></asp: Label>  </td>
</tr>
<tr>
    <td align=center valign=middle>
        <span style="color: navy">专业</span></td>
    <td><asp: Label ID="SPECIALTYLabel" runat="server"
        Text='<%# Eval("SPECIALTY") %>'></asp: Label>  </td>
    <td align=center valign=middle>
        <span style="color: navy">备注</span></td>
```

```
        <td colspan="3"><asp:Label ID="REMARKLabel" runat="server"
            Text='<%# Eval("REMARK") %>'></asp:Label>   </td>
    </tr>
        </table>
    </ItemTemplate>
```

在设计视图中将 DataList1 的 HorizontalAlign 改为“Center”，Width 改为“90%”。在智能标签的自动套用格式中选择“石板”。执行页面，界面如图 8-18 所示。

管理人员列表1

用户号	admin	姓名	admin	密码	admin
性别	男	生日	2001-1-1 0:00:00	职务	网络管理员
专业	计算机	备注			
用户号	gao	姓名	gao	密码	gao
性别	男	生日	1968-1-1 0:00:00	职务	职员
专业	计算机	备注			
用户号	li	姓名	li	密码	li
性别	男	生日	1973-1-1 0:00:00	职务	系主任
专业	院校管理	备注			
用户号	m	姓名	m	密码	m
性别	女	生日	2001-1-1 0:00:00	职务	网络管理员
专业	计算机	备注	ddd		
用户号	zhang	姓名	zhang	密码	zhang
性别	女	生日	1957-1-1 0:00:00	职务	院长
专业	院校管理	备注			

图 8-18　DataList 控件的执行效果(二)

8.3.3　对 DataList 控件编程

8.3.2 小节已经做到了以比较美观的形式对数据进行显示，但还不能对数据进行修改，因此还不能说已经能够进行数据管理。本小节继续完善该示例。

为了保留 8.3.2 小节的成果供参考，为网站再创建一个新的页面 ManagerManage2。ManagerManage2 是在 ManagerManage1 的基础上，继续增加对数据的修改功能。在应用实例中，功能相同的页面也是 ManagerManage2。

将 ManagerManage1 中<div>和</div>中间的代码复制到 ManagerManage2 中。将标题改为“管理人员列表 2”，此页面现在就能执行，效果与前面完全相同。

按 8.3.2 小节的方法，为数据源“生成 INSERT、UPDATE 和 DELETE 语句”。

利用 DataList 控件也可以完成数据项的就地编辑功能，效果甚至比 GridView 控件的编辑功能(不推荐使用)还要好，但需要付出更多的手工劳动。

首先要为 DataList 控件生成一个 EditItemTemplate，但至少在目前使用的 Visual Studio 版本中 EditItemTemplate 还不能自动生成。将 ItemTemplate 的内容整体复制到紧接着 ItemTemplate 的后面，编辑为 EditItemTemplate。将其中的 Label 控件改为使用 TextBox 控件，以实现输入。EditItemTemplate 模板的代码如下：

```
    <EditItemTemplate>
        <table border="1" width="100%">
```

```
    <tr>
        <td width="10%" align="center" valign="middle">
    <span style="color: navy">用户号</span></td>
        <td width="23%"><asp: Label ID="USERIDLabel1" runat="server"
    Text='<%# Eval("USERID") %>'></asp: Label></td>
        <td width="10%" align="center" valign="middle">
    <span style="color: navy">姓名</span></td>
        <td width="23%"><asp: TextBox ID="USERNAMETextBox" runat="server"
    Width="97%" Text='<%#
    Eval("USERNAME") %>'></asp: TextBox></td>
        <td width="10%" align="center" valign="middle">
    <span style="color: navy">密码</span></td>
        <td width="24%"><asp: TextBox ID="PASSWORDTextBox" runat="server"
    Width="97%" Text='<%#
    Eval("PASSWORD") %>'></asp: TextBox></td>
</tr>
<tr>
        <td align="center" valign="middle">
    <span style="color: navy">性别</span></td>
        <td><asp: TextBox ID="SEXTextBox" runat="server" Width="97%"
    Text='<%# Eval("SEX") %>'></asp: TextBox></td>
        <td align="center" valign="middle">
    <span style="color: navy">生日</span></td>
        <td><asp: TextBox ID="BIRTHDAYTextBox" runat="server" Width="97%"
    Text='<%# Eval("BIRTHDAY") %>'></asp: TextBox></td>
        <td align="center" valign="middle">
    <span style="color: navy">职务</span></td>
        <td><asp: TextBox ID="DUTYTextBox" runat="server" Width="97%"
    Text='<%# Eval("DUTY") %>'></asp: TextBox></td>
</tr>
<tr>
    <td align="center" valign="middle">
        <span style="color: navy">专业</span></td>
    <td><asp: TextBox ID="SPECIALTYTextBox" runat="server" Width="97%"
    Text='<%# Eval("SPECIALTY") %>'></asp: TextBox></td>
        <td align="center" valign="middle">
    <span style="color: navy">备注</span></td>
        <td colspan="3"><asp: TextBox ID="REMARKTextBox" runat="server"
    Width="98%" Text='<%# Eval("REMARK") %>' Rows="3"
    TextMode="MultiLine"></asp: TextBox></td>
    </tr>
    <tr><td colspan="6" align="center">
      <asp: Button runat="server"
          ID="ItemSaveButton"
```

```
                    Text="保存"
                    CommandName="update"/>
                     
                <asp:Button runat="server"
                    ID="ItemCancelButton"
                    Text="取消"
                    CommandName="cancel"/>
              </td></tr>
        </table>
</EditItemTemplate>
```

注意：除了用于输入的 TextBox 控件外，EditItemTemplate 中还包含两个按钮，它们的 CommandName 属性分别被设为“update”和“cancel”。

我们还需要提供一种能够从 ItemTemplate 进入编辑模式的途径。按照前面介绍的方法，可以在 ItemTemplate 的 table 中增加一行，在其中增加两个按钮，将“修改”按钮的 CommandName 属性设为“edit”，将“删除”按钮的 CommandName 属性设为“delete”。代码如下：

```
<tr>
    <td colspan="6" align="center">
    <asp:Button runat="server"
      ID="ItemEditButton"
      Text="修改"
      CommandName="edit"/>
       
    <asp:Button runat="server"
       ID="ItemDeleteButton"
       Text="删除"
       CommandName="delete"/>
    </td>
</tr>
```

为 DataList1 生成 EditCommand、DeleteCommand、UpdateCommand 和 CancelCommand 的事件处理函数并编码如下：

```
protected void DataList1_EditCommand(object source,
    DataListCommandEventArgs e)
{
    //确定当前选中的项
    DataList1.EditItemIndex=e.Item.ItemIndex;
    //重新绑定数据
    DataBind();
}
protected void DataList1_UpdateCommand(object source,
    DataListCommandEventArgs e)
```

```
{
    //取得修改后的值
    String USERID= ((Label)e.Item.FindControl("USERIDLabel1")).Text;
    String USERNAME= ((TextBox)e.Item.FindControl("USERNAMETextBox")).Text;
    String PASSWORD= ((TextBox)e.Item.FindControl("PASSWORDTextBox")).Text;
    String SEX= ((TextBox)e.Item.FindControl("SEXTextBox")).Text;
    String BIRTHDAY= ((TextBox)e.Item.FindControl("BIRTHDAYTextBox")).Text;
    String DUTY= ((TextBox)e.Item.FindControl("DUTYTextBox")).Text;
    String SPECIALTY= ((TextBox)e.Item.FindControl("SPECIALTYTextBox")).Text;
    String REMARK= ((TextBox)e.Item.FindControl("REMARKTextBox")).Text;

    //设置数据源的 Update 命令参数
    System.Web.UI.WebControls.Parameter param =
        SqlDataSource1.UpdateParameters["USERID"];
    param.DefaultValue= USERID;
    param= SqlDataSource1.UpdateParameters["USERNAME"];
    param.DefaultValue= USERNAME;
    param= SqlDataSource1.UpdateParameters["PASSWORD"];
    param.DefaultValue= PASSWORD;
    param= SqlDataSource1.UpdateParameters["SEX"];
    param.DefaultValue= SEX;
    param= SqlDataSource1.UpdateParameters["BIRTHDAY"];
    param.DefaultValue= BIRTHDAY;
    param= SqlDataSource1.UpdateParameters["DUTY"];
    param.DefaultValue= DUTY;
    param= SqlDataSource1.UpdateParameters["SPECIALTY"];
    param.DefaultValue= SPECIALTY;
    param= SqlDataSource1.UpdateParameters["REMARK"];
    param.DefaultValue= REMARK;
    //调用 SqlDataSource 的修改事件,修改数据库中的记录
    SqlDataSource1.Update();

    //将 EditItemIndex 属性值设为-1,从而退出编辑模式
    DataList1.EditItemIndex= -1;
    //重新绑定数据,以使数据更新
    DataBind();
}
protected void DataList1_CancelCommand(object source,
    DataListCommandEventArgs e)
{
    //将 EditItemIndex 属性值设为-1,从而退出编辑模式
    DataList1.EditItemIndex= -1;
    //重新绑定数据,以使数据更新
    DataBind();
```

```
}
protected void DataList1_DeleteCommand(object source,
     DataListCommandEventArgs e)
{
     //置删除命令的参数
     string recordID= (DataList1.DataKeys[e.Item.ItemIndex]).ToString();
     System.Web.UI.WebControls.Parameter param =
        SqlDataSource1.DeleteParameters["USERID"];
     param.DefaultValue= recordID;
     //调用 SqlDataSource 的删除事件,删除数据库中的记录
     SqlDataSource1.Delete();
     //重新绑定数据,以使数据更新
     DataBind();
}
```

代码中有较详细的注释,这里不再给出进一步的说明。

执行页面,界面如图 8-19 所示。

管理人员列表2

用户号	admin	姓名	admin	密码	admin
性别	男	生日	2001-1-1 0:00:00	职务	网络管理员
专业	计算机	备注			
修改 删除					

用户号	gao	姓名	gao	密码	gao
性别	男	生日	1968-1-1 0:00:00	职务	职员
专业	计算机	备注			
保存 取消					

图 8-19 DataList 控件的执行效果(三)

在显示的每条记录下面都增加了两个按钮。单击"删除"按钮删除当前记录;单击"修改"按钮,当前记录进入原地编辑状态,可对当前记录进行修改和保存。

8.3.4 进一步对 DataList 控件编程

到 8.3.3 小节为止,还不能对"管理人员列表"进行"新增"操作。

可以按照 8.3.3 小节介绍的方法:为页面增加一个"新增"按钮和一个 Panel 控件,当单击"新增"按钮时控制 Panel 控件可见并在其上输入新记录的数据。

8.3.3 小节介绍的方法,所有按钮事件的处理都在服务器端进行。服务器端的编程是重点,本书不准备对客户端的编程做特别详细的介绍。但也不希望给大家建立一种推荐大家尽量在服务器端编程的印象,事实上很多程序员更喜欢在客户端控制界面效果,以达到更快的响应速度。成熟的 Web 应用程序设计应该是服务器端编程和客户端编程相结合的,根据具体的需求,合理地运用各种技术。

为网站再创建一个新的页面 ManagerManage3。ManagerManage3 要在 ManagerManage2 的基础上,继续增加对数据的插入功能,在应用实例中,功能相同的页

面也是ManagerManage3。

将ManagerManage2中<div>和</div>中间的代码复制到ManagerManage3中，将ManagerManage2的隐藏程序代码也复制过来。将标题改为“管理人员列表3”。

在原有页面代码的后面(SqlDataSource1的后面)增加如下代码：

```
<table style="width: 100%;">
  <tr>
    <td align="center">
      <input id="btnAdd" type="button" value="新增"
          OnClick="btnAdd_Click()"/>
  </td>
  </tr>
  <tr id="inserttr" style="DISPLAY: none">
    <td align=center>
    <table border="1" style="color: #4A3C8C;background-color: #E7E7FF;"
        align=center>
    <tr>
        <td align="center" valign="middle" style="width: 71px">
            <span style="color: navy">用户号</span></td>
        <td width="23%"><asp: TextBox ID="USERIDTextBox" runat="server"
            Width="97%" Text=''></asp: TextBox></td>
        <td width="10%" align="center" valign="middle">
            <span style="color: navy">姓名</span></td>
        <td width="23%"><asp: TextBox ID="USERNAMETextBox" runat="server"
            Width="97%" Text=''></asp: TextBox></td>
        …
    </tr>
    …(略：其他字段的输入，可参考上一小节代码。)
    <tr><td colspan="6" align="center">
        <asp: Button ID="btnSave" runat="server" Text="保存"
            OnClick="btnSave_Click"/>
           
        <input id="btnCancel" type="button" value="取消"
            OnClick="btnCancel_Click()"/>
        </td></tr>
      </table>
    </td>
  </tr>
</table>
```

这是一个table控件。

table的第一行上是一个普通HTML按钮，其单击事件由客户端程序的btnAdd_Click()函数处理。

table的第二行被命名为inserttr，可由客户端程序控制是否显示。该行上是另一个

嵌套的 table，包含了新增记录的输入界面。内嵌 table 的最后一行包含两个按钮。“保存”按钮的处理过程需要写数据库，肯定要在服务器端处理，因此采用 ASP. NET 按钮。“取消”按钮只需要简单地将外层 table 的第二行置为不显示，可直接在客户端处理，因此使用普通 HTML 按钮。

两个普通 HTML 按钮的单击事件处理函数采用 JavaScript 编程，代码如下：

```
<script language="JavaScript" type="text/JavaScript">
function btnAdd_Click()
{
    //将输入区置为“显示”
    inserttr.style.display='block';
    //将各输入域置为空
    document.all.USERIDTextBox.value='';
    document.all.USERNAMETextBox.value='';
    document.all.PASSWORDTextBox.value='';
    document.all.SEXTextBox.value='';
    document.all.BIRTHDAYTextBox.value='';
    document.all.DUTYTextBox.value='';
    document.all.SPECIALTYTextBox.value='';
    document.all.REMARKTextBox.value='';
}

function btnCancel_Click()
{
    //将输入区置为“不显示”
    inserttr.style.display='none';
}
</script>
```

为“保存”按钮的单击事件编写服务器端处理函数如下：

```
protected void btnSave_Click(object sender, EventArgs e)
{
    //取得插入的新值
    String USERID=Request.Params["USERIDTextBox"].ToString();
    String USERNAME=Request.Params["USERNAMETextBox"].ToString();
    String PASSWORD=Request.Params["PASSWORDTextBox"].ToString();
    String SEX=Request.Params["SEXTextBox"].ToString();
    String BIRTHDAY=Request.Params["BIRTHDAYTextBox"].ToString();
    String DUTY=Request.Params["DUTYTextBox"].ToString();
    String SPECIALTY=Request.Params["SPECIALTYTextBox"].ToString();
    String REMARK=Request.Params["REMARKTextBox"].ToString();

    //置插入命令的参数
    System.Web.UI.WebControls.Parameter param =
```

```
        SqlDataSource1.InsertParameters["USERID"];
    param.DefaultValue= USERID;
    param= SqlDataSource1.InsertParameters["USERNAME"];
    param.DefaultValue= USERNAME;
    param= SqlDataSource1.InsertParameters["PASSWORD"];
    param.DefaultValue= PASSWORD;
    param= SqlDataSource1.InsertParameters["SEX"];
    param.DefaultValue= SEX;
    param= SqlDataSource1.InsertParameters["BIRTHDAY"];
    param.DefaultValue= BIRTHDAY;
    param= SqlDataSource1.InsertParameters["DUTY"];
    param.DefaultValue= DUTY;
    param= SqlDataSource1.InsertParameters["SPECIALTY"];
    param.DefaultValue= SPECIALTY;
    param= SqlDataSource1.InsertParameters["REMARK"];
    param.DefaultValue= REMARK;
    //调用 SqlDataSource 的插入事件,向数据库插入记录
    SqlDataSource1.Insert();

    //将 EditItemIndex 属性值设为-1,从而退出编辑模式
    DataList1.EditItemIndex=-1;
    //重新绑定数据,以使数据更新。
    DataBind();
}
```

执行页面,界面如图 8-20 所示。

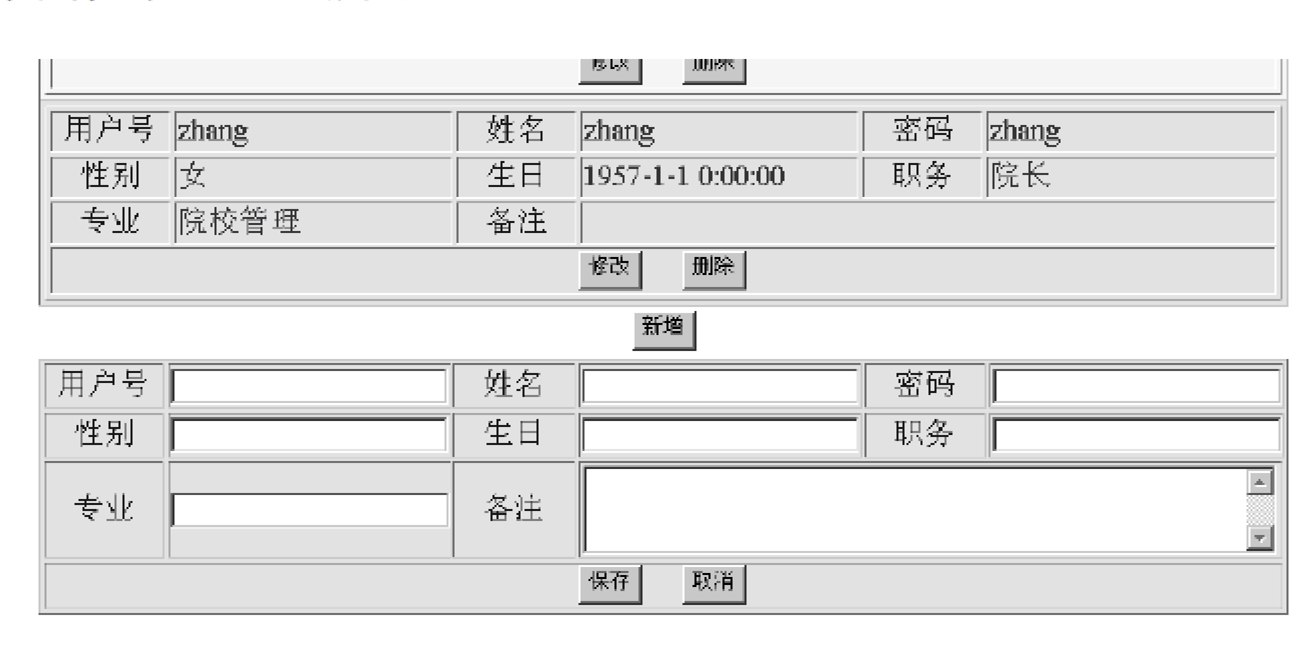


图 8-20 DataList 控件的执行效果(四)

与前面页面相比,页面的下部增加了一个“新增”按钮。单击“新增”按钮,可以看到,不需要经过页面的重新加载,页面的下部就出现了一个空白记录的输入区域。输入新记录信息,单击“保存”按钮,页面重新加载,新输入的记录出现在列表中。如果单击“取消”按钮,输入区直接消隐,页面不重新加载。

从本节的实例可以看出,使用 DataList 进行个性化的定制比使用 GridView 要容易。

但是,由于控件本身不提供内置的分页、排序等功能,因此,DataList 控件适合管理记录数不是太多的数据库表。

8.4 DetailsView控件

8.4.1 常用属性和事件

前面介绍的 GridView 控件和 DataList 控件都是以列表的形式一次显示多条记录。但在实际应用中经常需要一次只处理一条记录,如显示某类记录的细节信息,ASP. NET 提供了一些专门处理这种情况的控件,DetailsView 就是其中之一。

使用 DetailsView 控件可以显示数据库表中一条记录的信息,还可以执行编辑、删除和插入等操作。

DetailsView 控件与 GridView 有许多相同的属性,表 8-4 列出了 DetailsView 控件常用的属性和事件。

表 8-4 DetailsView 控件常用的属性和事件

属性名称	说明
AllowPaging	是否启用分页功能,默认为 False
AlternatingRowStyle	DetailsView 控件中的交替行的外观
AutoGenerateDeleteButton	记录中是否显示“删除”按钮,默认为 False
AutoGenerateEditButton	记录中是否显示“编辑”按钮,默认为 False
AutoGenerateInsertButton	记录中是否显示“插入”按钮,默认为 False
AutoGenerateRows	是否在 DetailsView 控件中为每个字段自动生成一行。注:类似于 GridView 控件的 AutoGenerateColumns 属性
BackColor、ForeColor、BorderColor、BorderStyle、BorderWidth 等	这些属性用于设置 DetailsView 控件的外观
BottomPagerRow	DetailsView 控件底部的分页行对象
CommandRowStyle	DetailsView 控件中的命令行的外观
CurrentMode	获取 DetailsView 控件的当前数据输入模式。有 3 个枚举项可选:Edit、Insert、ReadOnly。默认为 ReadOnly
DataItem	DetailsView 控件中的当前数据项
DataItemCount	数据源中的项数
DataItemIndex	数据源中当前项的索引,从 0 开始
DataKey	该属性值为一个 DataKey 对象,该对象表示所显示的记录的主键。注:GridView 控件中类似的属性为 DataKeys
DataKeyNames	该属性值为一个数组,该数组包含了显示在 DetailsView 控件中项的主键字段的名称
DataSource	该属性值为一个 DataSource 对象,DetailsView 控件从该对象中获得数据

续表

属性名称	说　明
DataSourceID	DetailsView 控件要绑定到的 DataSource 控件的 ID，DetailsView 从该控件中获得数据
DefaultMode	DetailsView 控件的默认数据输入模式。参 CurrentMode 属性
EditRowStyle	正在编辑的行的样式
FieldHeaderStyle	字段标题的样式
Fields	DetailsView 控件中所有字段的集合
HeaderRow	DetailsView 控件中的标题行对象
HeaderStyle	DetailsView 控件中的标题行的外观
HeaderTemplate	控件标题部分的模板
HeaderText	标题行中显示的文本
注：与上述 4 个属性相类似，还有与 EmptyData、Footer 相对应的属性	
InsertRowStyle	新插入的行的样式
PageCount	获取数据源中的记录数
PageIndex	当前记录的索引
PagerStyle	页导航行的样式
Rows	显示在 DetailsView 控件中的数据行 DetailsViewRow 对象的集合
SelectedValue	DetailsView 控件中的当前记录的数据键值
Visible	DetailsView 控件在页面上是否可见
事件名称	**说　明**
DataBinding	当 DetailsView 绑定到数据源时发生
DataBound	在 DetailsView 绑定到数据源后发生
ItemCommand	当单击 DetailsView 控件中的按钮时发生
ItemCreated	在 DetailsView 控件中创建记录时发生
ItemDeleted (ItemDeleting)	在单击 DetailsView 控件中的“删除”按钮时，但在删除操作之后(之前)发生
ItemInserted (ItemInserting)	在单击 DetailsView 控件中的“插入”按钮时，但在插入操作之后(之前)发生
ItemUpdated (ItemUpdating)	在单击 DetailsView 控件中的“更新”按钮时，但在更新操作之后(之前)发生
ModeChanged (ModeChanging)	在 DetailsView 控件试图在编辑、插入和只读模式之间更改时，但在更新 CurrentMode 属性改变之后(之前)发生
PageIndexChanged (PageIndexChanging)	当 PageIndex 属性的值在换页操作后(前)更改时发生

表 8-4 中的大部分属性和事件在前文的控件中都介绍过,这里不再赘述。

其中 DataItem 属性表示 DetailsView 控件中的当前数据项。通过编程对该属性进行操作,可直接当前记录各字段的值。

DetailsView 控件一次只显示一条记录。要想知道整个基础数据源中的记录数,使用 DataItemCount 属性。要想知道当前记录在整个基础数据源中位置,使用 DataItemIndex 属性。如果将 AllowPaging 属性设置为 True,这两个属性的值与 PageCount 属性和 PageIndex 的值相同。

若访问基础数据源中的所有记录,使用 Rows 属性。

8.4.2 DetailsView 控件的示例

从表 8-4 可以看出,DetailsView 控件可以触发许多与用户交互相关的事件,也就是说可以通过编程对数据库表中的单条记录实现丰富、细致的操作。本小节再给出一个使用 DetailsView 控件的简单示例。

创建一个名为 UseDetailsView 的网站。

为网站创建一个新的页面 TeacherManage1。最终要在此页面上完成对教师信息的管理。在本书的应用实例中,功能相同的页面也是 TeacherManage1。

为网页增加一个标题,代码如下:

```
<h1 align="center">教师管理 1</h1>
```

从工具箱拖一个 DetailsView 控件到页面上来,初始代码如下:

```
<asp:DetailsView ID="DetailsView1" runat="server" Height="50px"
    Width="125px">
</asp:DetailsView>
```

切换到设计视图,打开 DetailsView1 的智能标签,按照前文的方法,新建一个称为 SqlDataSource1 的数据源:连接字符串的名称用 SQLConnectionString,指定选取 TEACHER 表的所有字段,按 USERID 字段排序,为数据源“生成 INSERT、UPDATE 和 DELETE 语句”。

为 DetailsView1 创建了数据源后,在设计视图中可以看到其预览形式已经发生了变化——显示了记录的所有字段。将 DetailsView1 的宽度拖动到一个合适的值。回到源视图可以看到,除了增加了一个数据源对象外,DetailsView1 的代码也有较大的变化。代码如下:

```
<asp:DetailsView ID="DetailsView1" runat="server" Height="50px"
        Width="478px" AutoGenerateRows="False" DataKeyNames="USERID"
        DataSourceID="SqlDataSource1">
    <Fields>
        <asp:BoundField DataField="USERID" HeaderText="USERID"
            ReadOnly="True" SortExpression="USERID"/>
        <asp:BoundField DataField="USERNAME" HeaderText="USERNAME"
```

```
            SortExpression="USERNAME"/>
        <asp:BoundField DataField="PASSWORD" HeaderText="PASSWORD"
            SortExpression="PASSWORD"/>
        …(略:其他字段的绑定)
    </Fields>
</asp:DetailsView>
```

可以看出,DetailsView 各绑定字段的格式与 GridView 非常相似,可见这是两个封装程度非常相近的控件,由此也可以理解它们为什么有很多相同的属性。

执行页面,界面如图 8-21 所示。

在设计视图 DetailsView1 的智能标签中,将启用分页、启用插入、启用编辑和启用删除全部选中。在编辑字段对话框中将命令字段的 ButtonType 属性设为"Button"。自动套用"秋天"格式。将 DetailsView1 的 HorizontalAlign 属性设为"Center"。在源视图将字段标题改为中文。

再次执行页面,界面如图 8-22 所示。

教师管理1

USERID	gaoyi
USERNAME	高屹
PASSWORD	gaoyi
SEX	男
BIRTHDAY	1968-1-1 0:00:00
RANK	助教
SPECIALTY	计算机应用
REMARK	

图 8-21 DetailsView 控件的执行效果(一)

教师管理1

编号	gaoyi
姓名	高屹
密码	gaoyi
性别	男
出生日期	1968-1-1 0:00:00
职称	助教
专业	计算机应用
备注	
编辑 删除 新建	
1 2 3 4 5	

图 8-22 DetailsView 控件的执行效果(二)

从本节的示例可以看出,DetailsView 控件封装良好,具有完备的数据浏览与维护功能,基本不需要编程处理。

但是,由于数据记录之间的切换完全依靠分页导航来完成,因此只能适用于数据量很少,管理要求不高的情况。

功能上与 DetailsView 相似(同样是针对单个记录进行管理),但封装程度比 DetailsView 低(与 DataList 相似)的另一个控件是 FormView 控件。FormView 控件比较适合在主从关系的管理中维护细节信息,请参考应用实例中的"教师管理 2"功能。

习 题

1. 简述数据源控件和数据绑定控件的区别与联系。
2. DataSource 控件的作用是什么? ASP. NET 中包含哪些类型的 DataSource 控件?
3. 参照 8.1 节的介绍,自己动手实现 UseGridView 网站。

4. GridView 控件有什么功能?

5. 简述可以使用哪些方法控制 GridView 控件的外观。

6. 简述在 VS2005 的集成开发环境中,如何为 GridView1 的 RowDataBound 事件创建处理函数,并进入该函数的代码编辑。

7. 在习题 3 的基础上,使用 GridView 控件,通过操作 GridView 控件的属性和事件,分别实现 8.2 节所介绍的 7 个执行效果。

8. DataList 控件有什么功能?

9. DataList 与 GirdView 在显示数据的方式上有何异同?

10. DataList 控件都支持哪几类模板?

11. 参照 8.3 节的介绍,创建并实现其中的 UseDataList 网站,观察代码的执行效果,并与习题 7 中使用 GridView 控件的执行效果进行比较。

12. DetailsView 控件有什么功能? 在使用时有何局限性?

13. 参照 8.4 节的介绍,创建并实现其中的 UseDetailsView 网站,观察执行效果。

第 9 章 chapter 9

数据绑定

前两章介绍了 ASP.NET 数据库访问的两种方法。ADO.NET 方法是通过 ADO.NET 对象,使用 SQL 语句直接访问数据库;Web 数据访问是使用页面上的 Web 数据访问控件,通过封装的数据源控件访问数据库。

这两种方法各有优势,但单独使用在某些场合又会受到一定的限制。如:单独使用 ADO.NET 方法,数据库访问结果处理起来比较麻烦;单独使用 Web 数据访问,需要将 SQL 语句定义在页面中,不利于采用分层的方式编程。其实,可以将 ADO.NET 访问所得到的结果集(DataReader 或 DataSet)作为数据源绑定到 Web 控件上,这样就可以既得到 ADO.NET 数据访问的灵活性,又得到 Web 数据控件界面功能实现的简便性。

本章主要介绍数据绑定,也会介绍一些其他的数据使用及编码技巧。

9.1 嵌入式代码与简单数据绑定

9.1.1 嵌入式代码块

前面已经介绍了关于单文件页模型和代码隐藏模型的内容,详见 6.1.5 小节。这两种方式都需要编写独立代码,以交互的方式对控件属性进行控制,如改变控件的 Text 属性等。

除上述方法外,还可以使用嵌入式代码块将代码直接嵌入到网页中。嵌入式代码块是在页面加载的过程中执行的服务器代码。块中的代码可以包含编程语句,还可以使用当前页类中的成员变量、调用当前页类中的函数。嵌入式代码块必须使用页的默认语言进行编写,例如,如果页的@Page 指令中包含属性 Language="C#",则嵌入式代码块需要用 C#语言来编写。

嵌入式代码块的使用方法是:在页面代码中直接用<%和%>将相应语言的代码块括起来即可。

创建一个名为 EmbeddedCode 的网站。

在 Default.aspx 的<div>和</div>之间增加如下代码:

```
嵌入式代码块:<br>
<%
```

```
    for(int i=0; i<5; i++)
      Response.Write("<span style='font-size:"
        +(12+2*i).ToString()+"pt'>EmbeddedCode</span><br>");
%>
```

上述代码中使用嵌入式代码块，以不同的字体大小循环显示文本，执行效果如图 9-1 所示。

嵌入式代码块:
EmbeddedCode
EmbeddedCode
EmbeddedCode
EmbeddedCode
EmbeddedCode

图 9-1　嵌入式代码块的执行效果

ASP.NET 网页中支持嵌入式代码块，主要是与旧的 ASP 技术保持兼容。一般情况下，将嵌入式代码块用于复杂的编程逻辑并不是最佳做法，因为当页中的代码与标记混合时，很难进行调试和维护。此外，由于嵌入式代码块仅在页面加载时执行，因此其处理的灵活性比较低。

9.1.2　嵌入式表达式

如果不需要使用完整的代码块，还可以用<%=expression %>的形式，在网页中直接使用表达式的结果。其作用是：在页面加载时，将表达式的值直接插入到页面当中。

(1) 使用嵌入式表达式，可以取公共对象属性，如本地时间等。

在 Default.aspx 的<div>和</div>之间继续增加如下代码：

```
<br>使用表达式的结果:<br>
当前时间为:<%=DateTime.Now.ToString()%><br>
```

执行页面时将增加如下显示：

```
使用表达式的结果:
当前时间为:2007-6-18 11:18:52
```

(2) 使用嵌入式表达式，可以取页类成员变量的值。

在 Default.aspx.cs 中为页类_Default 增加一个成员变量：

```
protected string s1="protected string s1";
```

在 Default.aspx 的<div>和</div>之间继续增加如下代码：

```
<br>取页类的成员变量:<br>
<%=s1%><br>
```

执行页面时将增加如下显示：

```
取页类的成员变量:
protected string s1
```

(3) 使用嵌入式表达式，可以取成员函数的返回值。

在 Default.aspx.cs 中为页类_Default 增加一个成员函数：

```
protected string ReturnString()
```

```
{
    return "protected string ReturnString()";
}
```

该函数没有实际意义，只是直接返回一个字符串。

在 Default.aspx 的<div>和</div>之间继续增加如下代码：

```
<br>取成员函数的返回值：<br>
<%=ReturnString()%>< br>
```

执行页面时将增加如下显示：

```
取成员函数的返回值：
protected string ReturnString()
```

(4) 使用嵌入式表达式，可以取应用程序变量和会话变量的值。

在 Default.aspx.cs 中，将 Page_Load()函数改写为：

```
protected void Page_Load(object sender, EventArgs e)
{
    Application["ApplicationName"]="畅想网络学院";
    Session["UserName"]="admin";
}
```

在 Default.aspx 的<div>和</div>之间继续增加如下代码：

```
< br>取应用程序和会话变量的值：< br>
应用程序名称：<%=Application["ApplicationName"].ToString()%>< br>
用户名称：<%=Session["UserName"].ToString()%>< br>
```

执行页面时将增加如下显示：

```
取应用程序和会话变量的值：
应用程序名称：畅想网络学院
用户名称：admin
```

9.1.3 ASP.NET 表达式

嵌入式表达式在页面加载时将表达式的值直接插入到页面中，可插在页面的任何地方。如果仅仅是在页面加载时动态设置控件的属性，可以使用 ASP.NET 表达式。

使用 ASP.NET 表达式可将属性设置为连接字符串的值、应用程序配置项的值或资源文件中所包含的其他值。ASP.NET 表达式的基本语法如下：

```
<%$表达式前缀：表达式%>
```

美元符号($)表示后面所跟的是一个 ASP.NET 表达式。表达式前缀为 Web.config 文件中配置节的名称，如 AppSettings、ConnectionStrings 等。冒号后面的表达式部分是配置项的名称，在页面加载时将替换为该配置项的实际值。

使用 ASP.NET 表达式获得连接字符串，在前面章节其实已经多次使用，这里再做一个总结。按前面章节所述方法为网站创建 Web.config 文件，并增加适当的连接字符串 SQLConnectionString。

在 Default.aspx 的<div>和</div>之间继续增加如下代码：

```
<br>读配置文件：<br>
连接字符串为：<asp:Label ID="Label1" runat="server"
    Text="<%$ ConnectionStrings:SQLConnectionString%>"></asp:Label><br>
```

执行页面时将增加如下显示：

```
读配置文件：
连接字符串为：Data Source=(local);Initial Catalog=NetSchool;Persist Security Info=
True;User ID=sa;Password=abc
```

9.1.4 简单数据绑定

前面已经介绍了嵌入式表达式和 ASP.NET 表达式的用法，本小节再引入一种简单数据绑定表达式。

嵌入式表达式和 ASP.NET 表达式的值都是在页面加载时计算，而数据绑定表达式则是当调用页面的 DataBind 方法时才计算，并将所得值直接插入到页面当中去。

使用数据绑定表达式主要有两方面的功能，其中最常用的是与数据源绑定，其语法为：

```
<%#数据绑定表达式 %>
```

数据绑定表达式使用 Eval 和 Bind 方法将数据绑定到控件，并将更改提交回数据库。Eval 方法是只读方法，该方法采用字段名作为参数，以字符串的形式返回该字段的值。Bind 方法支持读/写功能，可以将被绑定控件值的更改提交回数据库。

其实在 8.3 节中介绍 DataList 控件的应用时，就已经大量使用了数据源绑定，如：

```
<asp:TextBox ID="USERNAMETextBox" runat="server" Width="97%"
    Text='<%#Eval("USERNAME") %>'>
</asp:TextBox>
```

其含义是：在执行数据绑定时，将 USERNAME 字段的值绑定到 ID 为"USERNAMETextBox"的 TextBox 控件上，作为其 Text 属性的值。

其实，除了与数据源绑定外，嵌入式表达式能够使用的值，数据绑定表达式也都可以使用，只不过绑定时机有所不同，本书将所有与非数据源的绑定称为简单数据绑定。

在 Default.aspx 的<div>和</div>之间继续增加如下代码：

```
<br>绑定其他控件属性：<br>
请输入文本：<asp:TextBox ID="TextBox1" runat="server"></asp:TextBox><br>
<asp:Button ID="Button1" runat="server" Text="绑定"
```

```
    OnClick="Button1_Click"/>
<asp:Label ID="Label2" runat="server" Text="<%# TextBox1.Text %>">
</asp:Label>
```

实现按钮的单击事件处理函数为：

```
protected void Button1_Click(object sender, EventArgs e)
{
    Page.DataBind();
}
```

执行页面，将增界面如图 9-2 的显示。

在文本输入框中输入任意文本，单击“绑定”按钮，页面重新加载，并将数据绑定，刚才输入的文本会显示在按钮之后。



图 9-2 简单数据绑定的执行效果

9.2 一般控件的数据绑定

在本书的 4.3 节中曾提到，如 ListBox、DropDownList 等控件都可以绑定到数据源。本节以 DropDownList 控件为例做一个具体介绍。将一般控件绑定到数据源有两种方法：一种是与 DataSource 对象绑定；另一种是绑定到 ADO.NET 的查询结果。

9.2.1 与 DataSource 对象绑定

创建一个名为 ControlBind 的网站。

为网站创建一个新的页面 TeacherManage2。最终要在此页面上，用一种新的方法完成对教师信息的管理。在应用实例中，功能相同的页面也是 TeacherManage2。本节只完成此功能的一部分。

为网页增加一个标题和一段文本：

```
<h1 align="center">教师管理 2</h1>
请选择教师：
```

从工具箱拖一个 DropDownList 控件到页面上来，切换到设计视图，打开 DropDownList1 的智能标签，选择“选择数据源”，打开“选择数据源”对话框。

在“选择数据源”列表中选择“新建数据源”，用前述的方法新建一个称为 SqlDataSource1 的数据源：连接字符串的名称用 SQLConnectionString，指定选取 TEACHER 表的 USERID 和 USERNAME 字段，按 USERID 字段排序。

如图 9-3 所示，为 DropDownList 控件指定要显示的字段和值字段。

数据源配置完成后，页面相关部分代码为：

```
<asp:DropDownList ID="DropDownList1" runat="server"
    DataSourceID="SqlDataSource1" DataTextField="USERNAME"
```

图 9-3 为 DropDownList 控件选择数据源

```
    DataValueField="USERID">
</asp:DropDownList>
<asp:SqlDataSource ID="SqlDataSource1" runat="server"
    ConnectionString="<%$ ConnectionStrings:SQLConnectionString %>"
    SelectCommand="SELECT [USERID], [USERNAME] FROM [TEACHER] ORDER BY
        [USERID]">
</asp:SqlDataSource>
```

执行页面，结果如图 9-4 所示，下拉式列表中列出了所有的教师姓名。



图 9-4 DropDownList 控件的执行效果

9.2.2 绑定到 ADO.NET 的查询结果

7.3 节介绍了使用 Command 对象和 DataReader 对象的方法。在该节的实例中，将数据库中读取的数据添加到 DropDownList 控件的列表项中，所使用的方法是调用 DropDownList 控件的 Items.Add()方法。下面用数据绑定的方法来完成同样的功能。

以 DropDownList 控件为例，绑定到 ADO.NET 的查询结果的一般方法为：

```
DropDownList控件.DataSource=数据源对象;
DropDownList控件.DataTextField=字段名;
DropDownList控件.DataValueField=字段名;
DropDownList控件.DataBind();
```

其中，数据源对象可以是 DataReader、DataSet 或 DataTable。字段名以字符串的方式提供。DataBind()是 DropDownList 控件的一个受保护的方法。

从工具箱中再拖一个 DropDownList 控件到页面上来。

在 Default.aspx.cs 的开头加上对 System.Data.SqlClient 命名空间的引用。

为_Default 类增加一个私有成员：

```
private string connectionString=
```

```
ConfigurationManager.ConnectionStrings["SQLConnectionString"].ConnectionString;
```

将 Page_Load()函数改为：

```
protected void Page_Load(object sender, EventArgs e)
{
    if (! Page.IsPostBack)
    {
        FillData();
    }
}
```

实现 FillData()函数如下：

```
private void FillData()
{
    SqlConnection conn= new SqlConnection(connectionString);

    string cmdText= "SELECT USERID,USERNAME FROM TEACHER ORDER BY USERID";
    SqlCommand command= new SqlCommand(cmdText, conn);

    try
    {
        //打开连接
        conn.Open();

        //执行查询
        SqlDataReader dr= command.ExecuteReader();

        //数据绑定
        DropDownList2.DataSource= dr;
        DropDownList2.DataTextField= "USERNAME";
        DropDownList2.DataValueField= "USERID";
        DropDownList2.DataBind();

        dr.Close();
    }
    catch (SqlException sqlex)
    {
        //显示错误信息
        Response.Write(sqlex.Message+ "<br>");
    }
    finally
    {
        //关闭数据库链接
        conn.Close();
```

```
    }
}
```

执行页面,可以看到两个 DropDownList 控件中具有完全相同的选项列表。

9.3 Web数据控件的数据绑定

第 8 章介绍了 Web 数据访问的有关内容,主要是 Web 数据控件通过 DataSource 控件来访问数据库。

其实,Web 数据控件也可以通过数据绑定的方式访问数据库,本节就以 GridView 控件为例进行介绍。将 GridView 控件绑定到 ADO.NET 数据源的一般方法如下:

```
GridView控件.DataSource=数据源对象;
GridView控件.DataBind();
```

其中,数据源对象可以是 DataReader、DataSet 或 DataTable。

在上述方法中没有指定字段如何绑定。与 8.2 节所介绍的方法相同,GridView 控件的字段绑定有两种方法:一种是将 AutoGenerateColumns 属性设为 True,系统会自动绑定;另一种是利用 BoundField 标签进行手工绑定。

创建一个名为 GridViewDataBind 的网站。网站的功能为:模仿 8.2.2 小节的功能,完成对学生信息的管理。

按前面章节所述方法为网站创建 Web.config 文件,并增加适当的连接字符串 SQLConnectionString。

为 Default.aspx 页面增加一个标题,代码如下:

```
<h1 class="title" style="text-align: center">学生管理 1</h1>
```

从工具箱中拖一个 GridView 控件到页面上来,按照 9.2 节的方法修改 Default.aspx.cs 文件中的代码,但 FillData()函数用下面代码来实现:

```
private void FillData()
{
    SqlConnection conn=new SqlConnection(connectionString);

    string cmdText="SELECT * FROM STUDENT";
    SqlCommand command=new SqlCommand(cmdText, conn);

    try
    {
        //打开连接
        conn.Open();

        //执行查询
        SqlDataReader dr=command.ExecuteReader();
```

```
            //数据绑定
            GridView1.DataSource=dr;
            GridView1.DataBind();

            dr.Close();
        }
        catch (SqlException sqlex)
        {
            //显示错误信息
            Response.Write(sqlex.Message+"<br>");
        }
        finally
        {
            //关闭数据库链接
            conn.Close();
        }
    }
```

其中粗体部分为 GridView1 的数据绑定代码。执行页面,效果如图 9-5 所示。

学生管理1

USERID	USERNAME	PASSWORD	SEX	BIRTHDAY	REGTIME	SPECIALTY	REMARK
200	张三	200	男	1989-8-1 0:00:00	2007-2-1 0:00:00	计算机	
201	s201	201	男		2007-3-24 17:33:28		
202	s202	202	男	2007-4-11 0:00:00	2007-4-5 0:00:00		
203	s203	203	男		2007-3-24 17:34:30	通信技术	
204	s204	204	男		2007-3-24 17:34:30		
205	s205	205	男		2007-3-24 17:34:30	数学	

图 9-5 GridView 控件的数据绑定(一)

将 GridView1 的 AutoGenerateColumns 属性置为“False”。为 GridView1 手工增加如下代码(也可从 8.2.2 小节的实例代码中复制过来再修改)。

```
<Columns>
    <asp:BoundField DataField="USERID" HeaderText="编号"/>
    <asp:BoundField DataField="USERNAME" HeaderText="姓名"/>
    <asp:BoundField DataField="PASSWORD" HeaderText="密码"/>
    <asp:BoundField DataField="SEX" HeaderText="性别"/>
    <asp:BoundField DataField="BIRTHDAY" HeaderText="出生日期"/>
    <asp:BoundField DataField="REGTIME" HeaderText="注册时间"/>
    <asp:BoundField DataField="SPECIALTY" HeaderText="专业"/>
    <asp:BoundField DataField="REMARK" HeaderText="备注"/>
</Columns>
```

为 GridView1 自动套用“石板”格式,执行页面,效果如图 9-6 所示。可以看到,页面的执行效果与前面很接近。

不幸的是,对于 GridView 这种封装良好的控件使用数据绑定方法,排序、分页、数据

学生管理1

编号	姓名	密码	性别	出生日期	注册时间	专业	备注
200	张三	200	男	1989-8-1 0:00:00	2007-2-1 0:00:00	计算机	
201	s201	201	男		2007-3-24 17:33:28		
202	s202	202	男	2007-4-11 0:00:00	2007-4-5 0:00:00		
203	s203	203	男		2007-3-24 17:34:30	通信技术	
204	s204	204	男		2007-3-24 17:34:30		
205	s205	205	男		2007-3-24 17:34:30	数学	

图 9-6　GridView 控件的数据绑定(二)

修改等功能都会受到影响。

但是,对于一些封装不像 GridView 这样完整的控件,如 DataList 等,使用数据绑定方法将获得更大的灵活性和更强的功能。特别应该指出的是:使用数据绑定方法有利于将应用程序分层实现,例如:将对数据库的存取操作放到数据库操作层,将应用逻辑放到逻辑层。

9.4 Repeater 控件

本节介绍 Repeater 控件的使用方法,从中读者可以更深刻地体会到如何使用数据绑定方法来获得更大的灵活性和更强的功能。

Repeater 控件是一个数据绑定容器控件,它可对数据源中的记录进行列表显示。Repeater 控件所提供的功能是 DataList 控件的一个子集:没有预定义的模板,没有提供列布局功能,没有提供对 Select/Edit/Delete 操作的支持。

很多程序员喜欢使用 Repeater 控件,因为它可以提供更强的灵活性。由于 Repeater 控件没有默认的外观,因此可以使用该控件创建许多种列表,如表(table)、逗号分隔的列表、XML 格式的列表等。

用户可以为 Repeater 控件定义如下的布局模板:

- ItemTemplate;
- AlternatingItemTemplate;
- HeaderTemplate;
- FooterTemplate;
- SeparatorTemplate。

各模板的含义与 DataList 的同名模板完全相同,这里不再赘述。但可以看到,与 DataList 控件相比,少了 SelectedItemTemplate 和 EditItemTemplate 模板。

Repeater 控件的公共属性也是 DataList 控件的一个子集。不但少了与选择和编辑相关的属性,还少了与外观相关的属性。也就是说:控制 DataList 控件的外观是通过指定 DataList 控件的外观属性来实现;而控制 Repeater 控件的外观则是采用传统的控制 HTML 样式的方法,这也许就是很多开发以前版本 ASP 应用程序的程序员喜欢使用 Repeater 控件的原因。

可以使用第 8 章中介绍过的方法,通过数据源控件,如 SqlDataSource,将 Repeater

控件绑定到数据源。本节主要介绍通过 ADO. NET 数据集(如 DataSet)、数据读取器(如 SqlDataReader)的数据绑定。

通过 ADO. NET 绑定数据时,首先需要为 Repeater 控件整体指定一个数据源。然后,需要向 Repeater 控件添加控件,包括显示控件(如 Label、TextBox)和控制控件(如 Button)等,并使用数据绑定语法(如<%# Eval("字段名") %>)将单个控件绑定到数据源返回的项的某个字段。

Repeater 控件支持多种事件,虽然它依然是 DataList 控件的一个子集,但已经很强大了。但是,使用 Repeater 控件时,程序员一般更倾向于自己来编程控制,请看下面的示例。

【例 9-1】 创建一个名为 UseRepeater 的网站。为网站创建 Web. config 文件,并增加连接字符串 SQLConnectionString。

为网站创建一个新的页面 Questions。最终要在此页面上完成提问与解答功能。提问与解答功能是"畅想网络学院"的核心功能之一,类似于一般网站的论坛。该功能针对每一门课程设置,学生和老师都可以就某一门课程提出问题(主题),又都可以对某个主题做出回答,从而展开讨论。页面 Questions 完成一门课程所有相关主题的分页列表,具有以下特点:

- 列表时需要显示每个主题的全部或部分内容,因此显示格式会比较复杂,显示内容可能需要进行预处理;
- 当用户单击某一主题的超链接时,可以显示该主题的细节及所有答复,还需要有"删除"等管理功能。

因此,该功能不适合用 GridView 等封装良好的控件完成,用 Repeater 这一类可以由用户灵活控制的控件特别合适。在应用实例中,功能相同的页面也是 Questions。

在应用实例中,Questions 页被其他页面所调用。在调用的过程中,将课程号作为参数传入,Questions 根据课程号显示相应课程的主题。本例进行了简化,假设只对课程号为"J03"的课程进行操作,因此,在 Questions. aspx. cs 文件中,为页类增加一个静态成员。

```
protected static string ClassID= "J03";
```

注意: 在应用实例中采用静态成员变量是为了能够在页面的多次加载之间保留数据,这样课程号只在第一次被调用时作为参数传入就可以了。

为网页增加一个标题,显示当前课程号。代码如下:

```
<h1 align= "center">问题与解答 (课号: <%=ClassID%>)</h1>
```

从工具箱拖一个 Repeater 控件到页面上来,初始代码如下:

```
<asp: Repeater ID= "Repeater1" runat= "server">
</asp: Repeater>
```

在上面两行间手工增加如下代码:

```
<HeaderTemplate>
```

```
<table width="100% " border="1">
    <tr>
      <td align="center" height="20" width="40% " bgcolor="#7A9BD1">
        <b>主题</b></td>
    <td align="center" height="20" width="15% " bgcolor="#7A9BD1">
        <b>作者</b></td>
    <td align="center" height="20" width="10% " bgcolor="#7A9BD1">
        <b>人气/回复</b></td>
    <td align="center" height="20" width="15% " bgcolor="#7A9BD1">
        <b>最新回复人</b></td>
    <td align="center" height="20" width="15% " bgcolor="#7A9BD1">
        <b>回复时间</b></td>
    <td align="center" height="20" width="5% " bgcolor="#7A9BD1">
        <b>删除</b></td>
</tr>
</HeaderTemplate>
<ItemTemplate>
<tr>
    <td align="left" bgcolor="#E7E7FF">
        <a href='Question_Res.aspx? BBSID=
            <%# DataBinder.Eval(Container.DataItem,"BBSID") %>'>
            <%# DataBinder.Eval(Container.DataItem, "TITLE")%>
        </a>
    </td>
    <td align="center" bgcolor="#E7E7FF">
        <%# DataBinder.Eval(Container.DataItem, "USERNAME")%></td>
    <td align="center" bgcolor="#E7E7FF">
        <%# DataBinder.Eval(Container.DataItem, "BBSREAD")%>/
        <%# DataBinder.Eval(Container.DataItem, "BBSWRITE")%></td>
    <td align="center" bgcolor="#E7E7FF">
        <%# DataBinder.Eval(Container.DataItem, "RESNAME")%>  </td>
    <td align="center" bgcolor="#E7E7FF">
        <%# DataBinder.Eval(Container.DataItem, "RESTIME")%></td>
    <td align="center" bgcolor="#E7E7FF">
        <a href='Questions.aspx? DelBBSID=
        <%# DataBinder.Eval(Container.DataItem,"BBSID") %>'>删除</a>
    </td>
</tr>
<tr>
    <td colspan="6" align="left" bgcolor="#E7E7FF">
        内容:<br>
            <span style="color: green"><%# DataBinder.Eval(Container.DataItem,
"CONTENT").ToString().Replace("\r\n", "</br>")%>  </span>
    </td>
```

```
            </tr>
        </ItemTemplate>
    <AlternatingItemTemplate>
        … (略:与 ItemTemplate 模板的代码基本相同,只是改变了背景颜色,如改为"#ffffcc")
</AlternatingItemTemplate>
<FooterTemplate>
    </table>
</FooterTemplate>
```

从上面代码可以看出:

- <HeaderTemplate>段用于显示表头。
- <ItemTemplate>和<AlternatingItemTemplate>用于交替显示各主题。
- 每个主题用 Table 的两行来显示。第一行显示基本信息,包括主题、作者、人气、最新回复人、回复时间和删除功能栏等;第二行用于显示内容。
- 用数据绑定表达式显示主题的各字段。
- 主题域被置为超链接,链接到该主题的答复页面 Question_Res. aspx,以主题号为参数。
- 删除域设置一个超链接,仍链接到本页面,但传递一个参数 DelBBSID,指明要删除的主题号。
- 内容在显示之前要进行预处理,将其中的回车换行符替换为</br>标记。

每门课程涉及的主题可能很多,因此需要进行分页显示。使用 Repeater 控件时,显示分页也需要用程序来主动控制。

为页类再增加一个静态成员和一个非静态成员,代码如下:

```
    protected static int pagenumber=1;
    private string connectionString=ConfigurationManager.
ConnectionStrings["SQLConnectionString"].ConnectionString;
```

在 Repeater 控件的后面增加一个 Label 控件,用于显示分页信息;再增加一个 DropDownList 控件,用于进行页导航。代码如下:

```
<asp:Label ID="Label1" runat="server" Text="Label"></asp:Label>
转到当前第<asp:DropDownList ID="DropDownList1" runat="server"
    AutoPostBack="True">
</asp:DropDownList>页
```

在 Questions. aspx. cs 的开头加上对 System. Data. SqlClient 命名空间的引用。

在 Page_Load()函数中增加数据绑定函数的调用代码:

```
//绑定数据
BindRepeater();
```

实现数据绑定函数的代码如下:

```
protected void BindRepeater()
```

```
{
    //创建连接对象
    SqlConnection conn=new SqlConnection(connectionString);
    DataTable dt=null;

    try
{
        //打开连接
        conn.Open();

        //填充 DataSet
        string cmdText="select * from BBS where CLASSID='"+ClassID
        +"' ORDER BY [RESTIME] DESC";
        SqlDataAdapter da=new SqlDataAdapter(cmdText, conn);
        DataSet ds=new DataSet();
        ds=new DataSet("bbs");
        da.Fill(ds, "bbs");
        dt=ds.Tables["bbs"];
    }
    catch (SqlException sqlex)
    {
        //显示错误信息
        Response.Write(sqlex.Message+"<br>");
    }
    finally
    {
        //关闭数据库链接
        conn.Close();
    }

    // 每页数据的条数
    int pagesize=10;
    // 记录数
    int recordcount=dt.Rows.Count;
    // 页数
    int pagecount=(recordcount-1)/pagesize+1;

    //当前页号
    if (DropDownList1.Text !="")
    {
        pagenumber=Convert.ToInt32(DropDownList1.Text);
    }

    int i;
```

```
    for (i=1; i<=pagesize*(pagenumber-1); i++)
        dt.Rows.RemoveAt(0);
    int j=dt.Rows.Count;
    for (i=pagesize; i<j; i++)
        dt.Rows.RemoveAt(pagesize);
    Repeater1.DataSource=dt;
    Repeater1.DataBind();

    string mess="共 <span style='color: red'>"
        +Convert.ToString(recordcount)
        +"</span>个问题  每页显示 <span style='color: red'>"
        +Convert.ToString(pagesize)
        +"</span>个问题共 <span style='color: red'>"
        +Convert.ToString(pagecount)+"</span>页          ";
    this.Label1.Text=mess;

    this.DropDownList1.Items.Clear();
    for (i=1; i<=pagecount; i++)
        this.DropDownList1.Items.Add(Convert.ToString(i));
    DropDownList1.SelectedIndex=pagenumber-1;

    dt.Clear();
}
```

代码中有较详细的注释,这里不再给出进一步的说明。

执行页面,如图 9-7 所示。

问题与解答（课号：J03）

主题	作者	人气/回复	最新回复人	回复时间	删除
ttt	教师1	0/0		2007-4-7 16:42:26	删除
内容： tttttttt tttttttttttttttttt tttt					
hhhhh	S201	1/0	t	2007-3-11 0:00:00	删除
内容： hhhhhhhhh					

共 12 个问题 每页显示 10 个问题 共 2 页 当前第 2 页

图 9-7 Repeater 控件的执行效果(一)

下面来实现页面的删除功能。

如果用户选择了删除某一个主题,仍然是调用本页面,但会传递一个要删除的主题号进来。因此,在 Page_Load()函数的开始部分先对此参数进行检测并处理。

```
//如果有删除命令,先处理删除
if (Request.Params["DelBBSID"]!=null)
{
    string DelBBSID=Request.Params["DelBBSID"].ToString();
    DeleteBBS(DelBBSID);
```

```
}
```

其中 DeleteBBS(DelBBSID)函数用于删除指定的主题,请读者参照 7.5 节的内容自行实现。

下面来为页面实现增加主题的功能。

在页面的下部增加一个 Panel 控件,上面包含发表新主题的界面,代码如下:

```
<asp:Panel ID="Panel1" runat="server" style="text-align: center"
     Width="100%">
   <table width="80%" border="1" bgcolor="linen"
     style="color: #4A3C8C;background-color: #E7E7FF;">
     <tr>
          <td colspan="2"><span style="font-size: 22px; color: green;
               font-family: 华文行楷">提出新问题</span></td>
     </tr>
     <tr>
          <td width="15%">
               <span style="color: navy">主题</span>
          </td>
          <td width="85%">
               <asp:TextBox ID="TITLETextBox" runat="server" Width="98%"
                    Text=''>
               </asp:TextBox>
          </td>
     </tr>
     <tr>
          <td>
               <span style="color: navy">内容</span>
          </td>
          <td>
               <asp:TextBox ID="CONTENTTextBox" runat="server"
                    Width="98%" Text='' Rows="4" TextMode="MultiLine">
               </asp:TextBox><br/>
          </td>
     </tr>
     <tr>
          <td colspan="2">
               <asp:Button ID="btnSave" runat="server"
                    OnClick="btnSave_Click" Text="发表"/>
          </td>
     </tr>
   </table>
</asp:Panel>
```

实现“发表”按钮的单击事件处理函数如下:

```
protected void btnSave_Click(object sender, EventArgs e)
{
    string USERTYPE="T";                    //假设当前用户类型为"教师"
    string USERID="teacherID";              //假设当前用户号为"teacherID"
    string USERNAME="teacherName";          //假设当前用户名为"teacherName"
    //插入主题
    InsertBBS(ClassID, TITLETextBox.Text, CONTENTTextBox.Text, USERTYPE, USERID, USERNAME);

    //重新绑定数据
    BindRepeater();

    //清除原输入内容
    TITLETextBox.Text="";
    CONTENTTextBox.Text="";
}
```

其中 InsertBBS()函数用于向数据库插入新的主题,请读者自行实现。

执行页面,效果如图 9-8 所示。

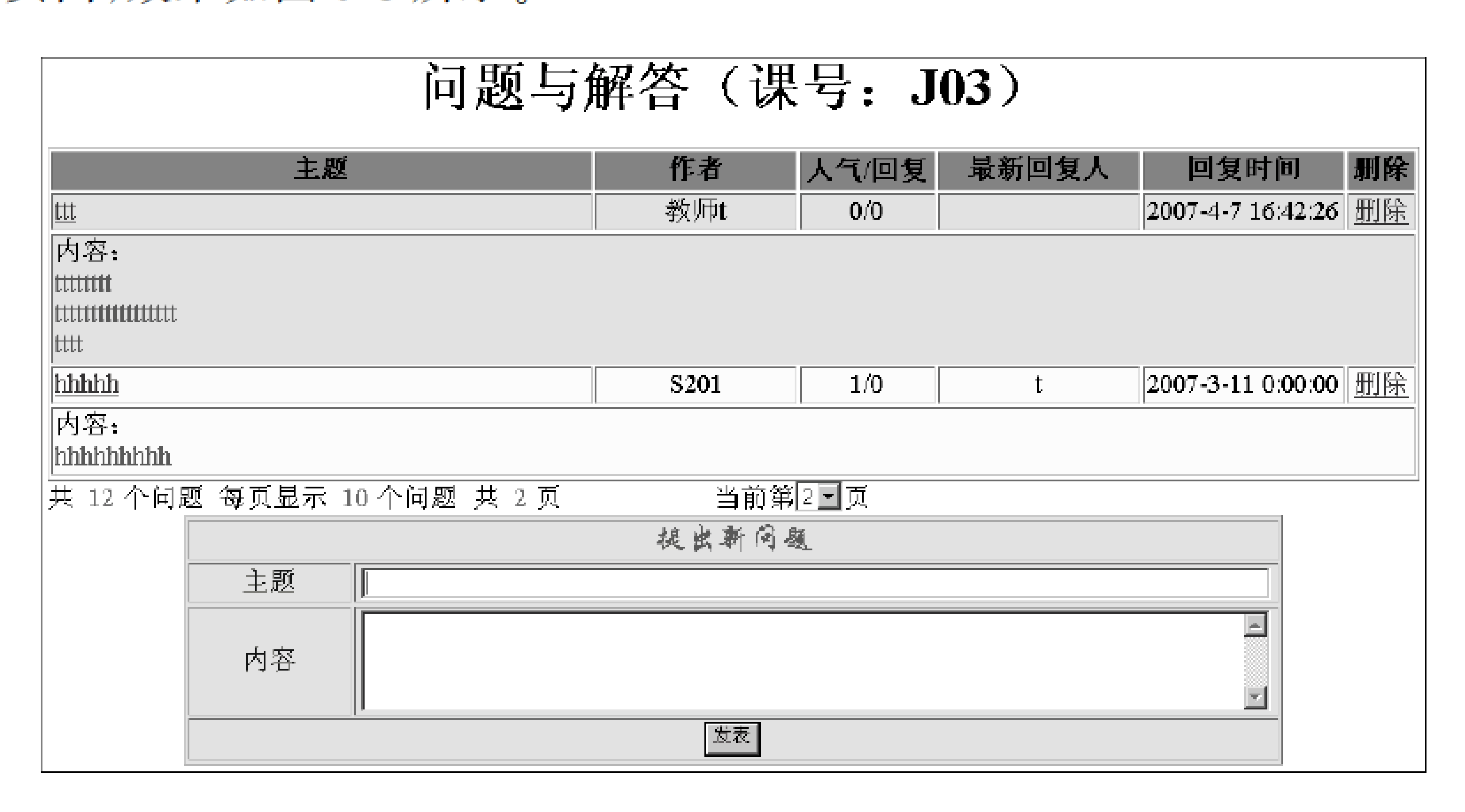


图 9-8 Repeater 控件的执行效果(二)

习　　题

1. 在 ASP. NET 中,引入数据绑定有何意义?
2. 在网页中如何使用嵌入式代码块?
3. 在网页中使用嵌入式表达式可以实现哪些功能?
4. ASP. NET 表达式有什么作用?
5. 请比较嵌入式代码块、嵌入式表达式和 ASP. NET 表达式的异同点。
6. 请说明嵌入式表达式、ASP. NET 表达式和数据绑定表达式在计算值时有何区别?

7. 请使用简单数据绑定的方法实现如图 9-2 所示的执行效果。

8. 采用哪些方法可以将一般控件绑定到数据源?

9. 分别使用 DataSource 对象绑定和绑定到 ADO. NET 的方法,实现 9.2.1 节介绍的 ControlBind 网站,并比较两种实现方法的执行效果。

10. Web 数据控件访问数据库有哪些方式?

11. 使用数据绑定的方法实现 9.3 节介绍的 GridViewDataBind 的网站。

12. 引入 Repeater 控件有何意义?

13. 参照 7.5 节内容,自行编写 9.4 节中所提到的 DeleteBBS(DelBBSID)函数,实现删除指定主题的功能。

14. 自行编写 9.4 节中所提到 InsertBBS()函数,实现向数据库插入新的主题的功能。

15. 请使用 Repeater 控件,实现 9.4 节介绍的 UseRepeater 的网站。

第 10 章 chapter 10

高级网站技术

在前面的章节中已经介绍了 Web 应用的基本开发技术。但是，要想构建并维护一个大型的专业网站，还需要用到一些非常重要的高级网站技术，这些高级网站技术至少包括母板页、导航、用户控件和部署等内容。

10.1 母 板 页

母板页用来创建统一的框架，每个页面（内容页）都可以把内容放在这个框架中，用来为应用程序创建统一的布局。

新创建的母板页原始代码如下：

```
<%@ Master Language="C#" AutoEventWireup="True"
    CodeFile="MasterPage.master.cs" Inherits="MasterPage" %>

<html xmlns="http://www.w3.org/1999/xhtml" >
<head runat="server">
    <title>无标题页</title>
</head>
<body>
    <form id="form1" runat="server">
    <div>
        <asp:contentplaceholder id="ContentPlaceHolder1" runat="server">
        </asp:contentplaceholder>
    </div>
    </form>
</body>
</html>
```

与一般页面很相似，同样可以包括静态文本、HTML 元素和服务器控件等内容，但有两个关键区别：

（1）母板页由特殊的@ Master 指令识别，见上述代码第 1 行，而普通.aspx 文件用@ Page 指令。

(2) 除一般 .aspx 页面可具有的内容外，母板页还可包括一个或多个 ContentPlaceHolder 控件，称为可替换内容占位符，见上述代码<div>标签之间的部分。在母板页中，ContentPlaceHolder 控件其实只定义可替换内容的名称，真正的内容要由各内容页提供。

应用程序中，要套用母板页中预定义布局的页面称为内容页。在内容页中，通过添加 Content 控件并将这些控件映射到母板页上的 ContentPlaceHolder 控件来创建内容，从而达到内容页与母板页的融合。之所以称为内容页，是因为它只需要提供母板页中“可替换内容”的内容，在内容页中，Content 控件之外的任何内容(除服务器代码的脚本块外)都将导致错误。

下面用一个示例说明如何在 Web 应用程序中使用母板页，如何为应用程序中的所有页面定义标准的外观和行为。

创建一个名为 UseMasterPage 的网站。为网站“添加新项”，选择“母板页”，接受默认的文件名 MasterPage. master。

可以看到，在母板页的<div></div>标记之间包含了<asp:contentplaceholder>声明：

```
<asp:contentplaceholder id="ContentPlaceHolder1" runat="server">
</asp:contentplaceholder>
```

将来，各内容页的内容会插入在<asp:contentplaceholder>声明之间；而在母板页中，可将网站通用的任何内容放置在该声明的周围。一个母板页中可以有多个<asp:contentplaceholder>，每个都有其唯一的、可具有实际含义的 id。

将母板页的代码修改为(不包括每行前面的行号)：

```
<%@ Master Language="C#" AutoEventWireup="True"
CodeFile="MasterPage.master.cs" Inherits="MasterPage" %>

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">

<script language="JavaScript" type="text/JavaScript">
    window.moveTo(0,0);
    window.resizeTo(screen.width,screen.height-25);
</script>

<html xmlns="http://www.w3.org/1999/xhtml" >
<head runat="server">
    <title>无标题页</title>
<script language="JavaScript" type="text/JavaScript">
function Close()
{
    top.opener=null;
```

```
16:     top.window.close();
17: }
18: </script>
19: </head>
20: <body leftmargin=0 rightmargin=0 topmargin=0 bottommargin=0
21: background="images/39.jpg">
22:     <div>
23:         <table width=100% height="70" background=images/title_20.jpg
24: cellpadding="0" cellspacing="0" align=center>
25:             <tr>
26:                 <td valign="top" width=800 background=images/title0.jpg>
27:                     <div id="Div1" style="position:absolute; width:40px;
28: height:20px; z-index:1; left: 720px; top: 30px;cursor:hand;"
29:                     title="关闭" onclick="Close()"></div>
30:                 </td>
31:                 <td> </td>
32:             </tr>
33:         </table>
34:     </div>

35:     <form id="form1" runat="server">
36:     <div>
37:         <asp:contentplaceholder id="ContentPlaceHolder1" runat="server">
38:         </asp:contentplaceholder>
39:     </div>

40:     <div align="center">
41:         <span style="color: red">网站统一的版权信息</span>
42:     </div>
43:     </form>
44: </body>
45: </html>
```

其中：

1～4 行为母板页的原始代码，由系统自动生成。

5～8 行为 JavaScript 代码，在页面加载到客户端时执行，功能为改变浏览器窗口的位置和大小。使用该母板页的所有内容页都会执行这部分代码。

22～34 行：显示在内容页上部的内容。本例显示了一个背景图片，并在其上某个特定的位置(对应于图片上“关闭”处)定义了一个层<div id="Div1">，当单击该层时执行 JavaScript 代码中的 Close()函数。

12～18 行：包含 Close()函数的 JavaScript 代码。

40～42 行：显示在内容页下部的内容，可以是网站统一的版权信息等内容。

为网站添加一个新页 Page1.aspx。需要注意的是，在“添加新项”对话框中需要选中

右下部的“选择母板页”对话框。

按“添加”按钮后，在“选择母板页”对话框中选择 MasterPage. master。用这种方法添加的新页，初始代码为：

```
<%@ Page Language="C#" MasterPageFile="~/MasterPage.master"
    AutoEventWireup="True" CodeFile="Page1.aspx.cs" Inherits="Page1"
    Title="Untitled Page" %>
<asp:Content ID="Content1" ContentPlaceHolderID="ContentPlaceHolder1"
    Runat="Server">
</asp:Content>
```

可以看出代码与一般的页面大不相同。

在内容页的@ Page 指令中，通过将 MasterPageFile 属性指向要使用的母板页，实现内容页与母板页的绑定。

在第一行中将“Title="Untitled Page"”改为“Title="第 1 个页面"”，这将影响本页面标题栏中的显示。

在<asp:Content>标记内增加如下代码：

```
这是第 1个页面，按
<asp:HyperLink ID="HyperLink1" runat="server" NavigateUrl="Page2.aspx">
    这里
</asp:HyperLink>
可打开第 2个页面。
```

用上述同样的方法为网站再添加一个新页 Page2. aspx。将其标题栏标题改为“第 2 个页面”，在<asp:Content>标记内增加如下简单代码。

```
这是第 2个页面。
```

执行第 1 个页面，界面如图 10-1 所示。



图 10-1 使用了母板页的页面(一)

单击超链接打开第 2 个页面，界面如图 10-2 所示。

可以看到，两个窗口具有同样的外观风格和操作风格。在两个窗口中单击“关闭”按钮，都可以关闭窗口。无论将浏览器窗口的大小改为多少，重新打开这两个页面的任何



图 10-2 使用了母板页的页面(二)

一个时都会将窗口重新置为充满屏幕。

10.2 导　　航

互联网才开始流行时，网站都比较简单，只需要使用一般的超链接就能完成站内的导航任务。当时有一种说法，就是认为摒弃 C/S 结构应用程序的菜单是一个进步。

现在，网站越来越大，内容越来越丰富，很多网站又不得不增加了菜单式的导航方式。不幸的是，菜单并不是 HTML 的原始特色，但这却给程序员提供了发挥的空间。他们用 HTML 与脚本语言相结合，做出了多种风格的漂亮菜单，有的封装良好，甚至可以当成控件来使用。但问题是这些控件没有统一的接口，编程人员要想继承和使用这些控件需要做大量的学习与编码修改工作。

ASP.NET 提供的 TreeView 控件具有树形外观，特别适合作为菜单或导航工具。本书的应用实例中就采用了 TreeView 控件生成菜单，所使用的方法是：从数据库中读出各菜单项，由程序控制生成 TreeView 菜单。本节介绍另一种方法：通过特定的 XML 文件来生成 TreeView 各节点的方法。

ASP.NET 还提供了专用的导航控件，如 SiteMapDataSource、SiteMapPath 等，将它们与 TreeView 控件结合使用，可以达到很好的导航效果，且编程更简单，请看下面的实例。

创建一个名为 UseSiteMap 的网站。

为网站"添加新项"，选择"站点地图"，接受默认的文件名 Web.sitemap，其原始代码为：

```
<?xml version="1.0" encoding="utf-8" ?>
<siteMap xmlns="http://schemas.microsoft.com/AspNet/SiteMap-File-1.0" >
    <siteMapNode url="" title=""  description="">
        <siteMapNode url="" title=""  description=""/>
        <siteMapNode url="" title=""  description=""/>
    </siteMapNode>
</siteMap>
```

这是一个 XML 文件。其中每个 siteMapNode 对应 TreeView 中的一个节点，可以嵌套。每个 siteMapNode 包含 3 个属性，其中 title 为节点的标题，当鼠标移动到该节点上时，显示 description 的内容作为提示信息。

将 Web. sitemap 的内容改为：

```
<?xml version="1.0" encoding="utf-8" ?>
<siteMap xmlns="http://schemas.microsoft.com/AspNet/SiteMap-File-1.0" >
    <siteMapNode url="manager.aspx" title="管理人员系统"
        description="供管理人员使用。">
        <siteMapNode url="teachermanage.aspx" title="教师管理"
                description="对教师信息进行管理。">
            <siteMapNode url="teachermanage1.aspx" title="教师管理 1"
                description="使用 DetailsView 控件。"/>
            <siteMapNode url="teachermanage2.aspx" title="教师管理 2"
                description="使用 FormView 控件。"/>
        </siteMapNode>
        <siteMapNode url="studentmanage.aspx" title="学生管理"
            description="对学生信息进行管理。"/>
        <siteMapNode url="classmanage.aspx" title="课程管理"
            description="对课程信息进行管理。"/>
    </siteMapNode>
</siteMap>
```

本示例还希望网站的主要页面具有统一的布局风格，如使用如图 10-3 所示的布局，为此需要使用母板页。

<table>
<tr><td rowspan="2">统一的
站点导航</td><td>站点路径</td></tr>
<tr><td>各页的具体内容。</td></tr>
</table>

图 10-3　页面的统一布局

为网站添加一个母板页，接受默认的文件名 MasterPage. master。

为母板页增加一个 table 来实现上述布局，为了在内容页中能清楚地看出布局，将 table 的 border 属性置为 1。

```
<table Width="100% " border="1">
    <tr>
        <td style="width: 30% " rowspan="2">
        </td>
        <td style=" width: 70% ;height: 28px">
        </td>
```

```
        </tr>
        <tr>
            <td>
            </td>
        </tr>
    </table>
```

将<asp:contentplaceholder>声明移入右下的单元格中，将来在这里显示各页的具体内容。

在设计视图中，从工具箱拖一个 SiteMapDataSource 和一个 TreeView 移到左侧的单元格中。

SiteMapDataSource 控件会自动搜索并使用默认的站点地图(Web. sitemap)，不需要再对其进行特别的配置。

打开 TreeView1 的智能标签，在“选择数据源”中选择 SiteMapDataSource1，之后可以看到 TreeView 控件外观的改变，已经包含了实际执行效果的预览。

拖一个 SiteMapPath 移到右上角的单元格中，暂时不用对其进行任何设置。

完成上述修改后，MasterPage. master 相关部分的代码改变如下：

```
<table Width="100% " border="1">
    <tr>
        <td style="width: 30% " rowspan="2">
            <asp:SiteMapDataSource ID="SiteMapDataSource1"
                runat="server"/>
            <asp:TreeView ID="TreeView1" runat="server"
                DataSourceID="SiteMapDataSource1">
            </asp:TreeView>
        </td>
        <td style=" width: 70% ;height: 28px">
            <asp:SiteMapPath ID="SiteMapPath1" runat="server">
            </asp:SiteMapPath>
        </td>
    </tr>
    <tr>
        <td>
            <asp:contentplaceholder id="ContentPlaceHolder1"
                runat="server">
            </asp:contentplaceholder>
        </td>
    </tr>
</table>
```

为网站增加页面，包括 manager. aspx、teachermanage. aspx、teachermanage1. aspx、teachermanage2. aspx 等，并为其添加适当的内容。注意在添加页面时需要选中“选择母板页”复选框。

执行系统，当打开“教师管理 1”页面时，如图 10-4 所示。

管理人员系统
教师管理
教师管理1
教师管理2
学生管理
课程管理
管理人员系统 > 教师管理 > 教师管理1
教师管理1
使用DetailsView控件。

图 10-4 使用了导航的页面(一)

可以通过修改 TreeView 控件和 SiteMapPath 控件的属性来改变其外观。TreeView 所包含的属性和事件非常多，表 10-1 中只列出了其中最常用的一小部分，仅作为提示。

表 10-1 TreeView 控件的重要属性和事件

属性名称	说明
BorderStyle、Height 等	控制 TreeView 控件的外观
CheckBoxes	是否在 TreeView 控件中的树节点旁显示复选框
LabelEdit	是否可以编辑树节点的标签文本
Nodes	TreeView 控件的节点集合
Scrollable	TreeView 控件是否在需要时显示滚动条
SelectedNode	当前在 TreeView 控件中选定的节点
ShowNodeToolTips	当鼠标指针悬停在 TreeNode 上时显示的提示
Sorted	TreeView 中的树节点是否经过排序
事件名称	**说明**
CollapseAll/ExpandAll	折叠/展开所有树节点
GetNodeAt	检索位于指定位置的树节点
GetNodeCount	检索分配给 TreeView 控件的树节点数

也可以直接选用 VS2005 提供的格式：在母板页 MasterPage.master 的设计视图中，打开 TreeView1 的智能标签，自动套用“简明型”格式。再打开 SiteMapPath1 的智能标签，也自动套用“简明型”格式。将 SiteMapPath1 的 PathSeparator 属性改为“→”。再执行系统，效果如图 10-5 所示。



图 10-5 使用了导航的页面(二)

10.3 用户控件

10.3.1 用户控件的使用

用户可以将ASP.NET页面的一部分单独保存起来，然后在多个ASP.NET页面中重复使用，这单独保存起来的一部分页面称为用户控件。一般页面(.aspx)中的内容都可以用于用户控件。一般页面与用户控件也存在一些不同，包括：

- 用户控件文件的扩展名为.ascx；
- 用户控件中的HTML标记体系不必完整，可以不包括<html>、<body>等标记；
- 用户控件的开头是一个@ Control指令，而不是@ Page指令。

最简单的用户控件可以只显示静态内容，例如，可以将应用程序的版权信息做成一个用户控件。下面就以版权信息控件为例，说明用户控件的创建与使用方法。

1. 创建用户控件

创建一个名为UseUserControls的网站。

为网站“添加新项”，选择“Web用户控件”，将文件名改为copyright.ascx，其原始代码只有一行：

```
<%@ Control Language="C#" AutoEventWireup="True" CodeFile="copyright.ascx.cs" Inherits="
copyright" %>
```

可以在该行代码下面添加任何页面内容。copyright是一个通用的版权信息用户控件，只需要为其增加简单的文本内容即可：

```
<center><span style="color: tomato; font-size: 10pt;">
    如果您是出于学习目的,可以自由使用本系统的所有代码。…(略)
</span></center>
```

2. 使用用户控件

用户控件的使用需要遵循“先声明，后引用”的原则。

要在一个页面中使用某个用户控件，需要先使用@ Register指令来声明该用户控件，其语法格式为：

```
<%@ Register tagprefix="tagprefix" tagname="tagname" src="pathname" %>
```

tagprefix(标记前缀)和tagname(标记名)是用户定义的标识符。src属性值指明用户控件文件(.ascx)的位置，既可以是相对路径，也可以是从应用程序的根目录到用户控件源文件的绝对路径。例如，如果要声明前面创建的copyright用户控件，只需要在页面代码中增加如下代码：

```
<%@ Register TagPrefix="uc8" TagName="copyright" Src="copyright.ascx" %>
```

用户控件声明之后就可以引用了，其基本语法格式如下：

```
<TagPrefix:TagName runat="server"/>
```

在引用用户控件时还可以指定 ID 等属性，如：

```
<uc8:copyright ID="copyright" runat="server"/>
```

TagPrefix 和 TagName 属性总是以冒号分隔对（TagPrefix：TagName）的形式一起使用，构成完整的服务器标记。与一般控件的引用相比较，如：

```
<asp:Button ID="Button1" runat="server" Text="Button"/>
```

可以看出：TagPrefix 相当于 asp，因此一般在定义时也使用比较简练的标识符。TagName 相当于控件的类名（如 Button），因此一般使用能确实表明控件特性的标识符。但这只是一种编程习惯，为了使代码更清晰。需要再次说明，TagName 使用用户自定义的标识符，不需要创建控件时的类名。

10.3.2 NewsUC.ascx 用户控件

10.3.1 小节中所使用的用户控件 copyright 非常简单。其实用户控件中可以包含复杂的应用逻辑，可以像一般页面一样进行服务器端编程。下面以应用实例中所使用的新闻用户控件为例进行说明。

为网站再创建一个用户控件 NewsUC.ascx，该控件用于显示学院新闻，应用实例中功能相同的用户控件也是 NewsUC.ascx。为新创建的用户控件增加如下代码：

```
<table width="214" border="0" cellpadding=0 cellspacing=0>
    <tr>
        <td colspan="3"><img src="images/56.jpg" width="214px"/></td>
    </tr>
    <tr>
        <td style="width: 12px"><img src="images/57.jpg" width="12"/></td>
        <td background="images/58.jpg" width=192xp>
            <marquee id="ScrollNews" scrollamount="1" scrolldelay="50"
                direction="up" behavior="scroll"
                height=100px onmousemove="stop()" onmouseout="start()">
                <asp:DataList ID="NewsList" runat="server" Width="190px">
                    <ItemTemplate>
                        <a href='../Admin/Information/NewsInfo.aspx?
NewsID=<%#DataBinder.Eval(Container.DataItem,"NewsID") %>' target="_blank">
<%#DataBinder.Eval(Container.DataItem,"TITLE") %></a>
                    </ItemTemplate>
                    <ItemStyle HorizontalAlign="Center"/>
                </asp:DataList>
```

```
            </marquee>
        </td>
        <td width="12xp"><img src="images/59.jpg" width="12"/></td>
    </tr>
    <tr>
        <td colspan="3" align="left"><img src="images/60.jpg"
            width="214"/></td>
    </tr>
</table>
```

对上述代码的主要部分说明如下：

- 使用 DataList 控件显示新闻标题，绑定数据源的 TITLE 字段。
- 每个标题被设置为一个超链接，链接到新闻的具体内容，需要与数据源的 NewsID 字段绑定。
- 使用 marquee 标签将 DataList 包含其中。Marquee 为滚动文本选取框控件，详细的说明请参阅 MSDN。

打开该控件的隐藏代码文件 NewsUC. ascx. cs，可以看到其原始代码与一般页面的原始代码完全相同，也包含 Page_Load()函数。

请读者自行为 Page_Load()函数编写代码，从数据库中读取数据，作为数据源绑定到 NewsList 控件。可参阅 8.3 节的实例代码，SQL 命令用“select top 10 * from news order by NewsDate desc”。

10.3.3 ActiveOp.ascx 用户控件

除了与一般页面相同的服务器端处理之外，用户控件还包含与一般页面相同的客户端脚本程序。下面就以应用实例中所使用的一个活动操作控件为例进行说明。

为网站再创建一个用户控件 ActiveOp. ascx。该控件包含“学校公告”和“友情链接”等几部分内容，但为了节省页面空间，同一时刻只有一部分是激活的，其他部分只显示标题；用户单击哪一部分的标题，该部分就激活并显示。本书应用实例中功能相同的用户控件也是 ActiveOp. ascx。

将 ActiveOp. ascx 的代码修改为：

```
<%@  Control Language="C#" AutoEventWireup="True" CodeFile="ActiveOp.ascx.cs" Inherits="
ActiveOp" %>

<script language="JavaScript" type="text/JavaScript">
function showsubmenu(sid)
{
    eval('submenu' +1 + '.style.display=\'none\';');
    eval('submenu' +2 + '.style.display=\'none\';');
    eval('submenu' +sid + '.style.display=\'block\';');
}
```

```
</script>

<table width="210" border="1" align="center" cellPadding="0" cellSpacing="0" style="BORDER-
COLLAPSE: collapse">
    <tr>
        <td onclick="showsubmenu('1')">
            <img src="images/gonggao.jpg" width=210/>
        </td>
    </tr>
    <tr id="submenu1" style="DISPLAY:block">
        <td align="left" valign="top">
            畅想网络学院已正式开学,欢迎各位同学试用。<br/>
        </td>
    </tr>
    <tr>
        <td onclick="showsubmenu('2')">
            <img src="images/youqing.jpg" width=210/>
        </td>
    </tr>
    <tr id="submenu2" style="DISPLAY:none">
        <td align="center" valign="top">
            <table width="100%" border="0" cellpadding="0" cellspacing="0">
              <tr>
                <td width="50%" height="80" align="center">
                    <img src="images/ilc.gif" width="88" height="60"><br>
                    <a href="http://www.ilc.net.cn" target="_blank">我爱单车</a>
                </td>
                <td width="50%" height="80" align="center">
                    <img src="images/jsbike.gif" width="88" height="60"><br>
                    <a href="http://www.jsbike.com.cn" target="_blank">江苏单车网</a>
                </td>
              </tr>
            </table>
        </td>
    </tr>
</table>
```

上述代码说明如下:

- 使用一个 table 来完成布局。
- 每一部分占 table 的两行,一行为标题,一行为内容。
- 各部分的内容行都指定了 DISPLAY 属性,只有一部分的 DISPLAY 属性为 block(显示),其他各部分都为 none(不显示)。
- 为标题行的单元格指定 onclick 事件处理函数为 showsubmenu()。
- 在 showsubmenu()函数中将当前部分的内容行置为显示,其他各部分的内容行

都置为不显示。

在网站的默认页 Default.aspx 中声明如下三个用户控件：

```
<%@ Register TagPrefix="uc6" TagName="NewsUC" Src="NewsUC.ascx" %>
<%@ Register TagPrefix="uc5" TagName="ActiveOp" Src="ActiveOp.ascx" %>
<%@ Register TagPrefix="uc8" TagName="copyright" Src="copyright.ascx" %>
```

然后可对其进行引用：

```
<h1>使用用户控件</h1>
<uc6:NewsUC id="MyNewsUC" runat="server"></uc6:NewsUC>
<uc5:ActiveOp ID="ActiveOp" runat="server"/>
<uc8:copyright ID="copyright" runat="server"/>
```

执行页面，界面如图 10-6 所示。



图 10-6 使用了用户控件的页面

10.4 网站的部署

网站的部署不是本书的重点内容，但为了保证内容的完整性，本书还是简单介绍一下有关部署的内容。

网站的部署是将网站安装到其他 Web 服务器上的过程。Visual Studio 2005 提供两种不同的部署策略：ClickOnce 部署和 Windows Installer 部署。使用 ClickOnce 时，要使用发布向导打包应用程序，并将其发布到网站或网络文件共享；用户直接从该位置一次性地安装和启动应用程序。通过 Windows Installer 部署，则是将应用程序打包到 setup.exe 文件中，并将该文件分发给用户，用户可以运行 setup.exe 文件安装应用程序。

本节只通过一个实例介绍采用 Windows Installer 策略，为 Web 网站制作安装程序的最简单步骤。忽略了其中很多细节，更多的内容请参考 MSDN。

先来创建一个名为 WebAppSetup 的简单网站。

为 Default. aspx 增加一段文本内容：

```
网站安装实例：如果您看到了这段文字，说明网站安装已经成功。
```

前文介绍过，VS2005 在创建网站项目的同时，系统也为该网站创建了一个（一般情况下是同名的）解决方案。

在解决方案资源管理器中的解决方案名上单击右键，选择"添加"→"新建项目"。在"添加新项目"对话框中选择"其他项目类型"→"安装和部署"→"Web 安装项目"，接受默认的名称，选择正确的文件位置，如图 10-7 所示。

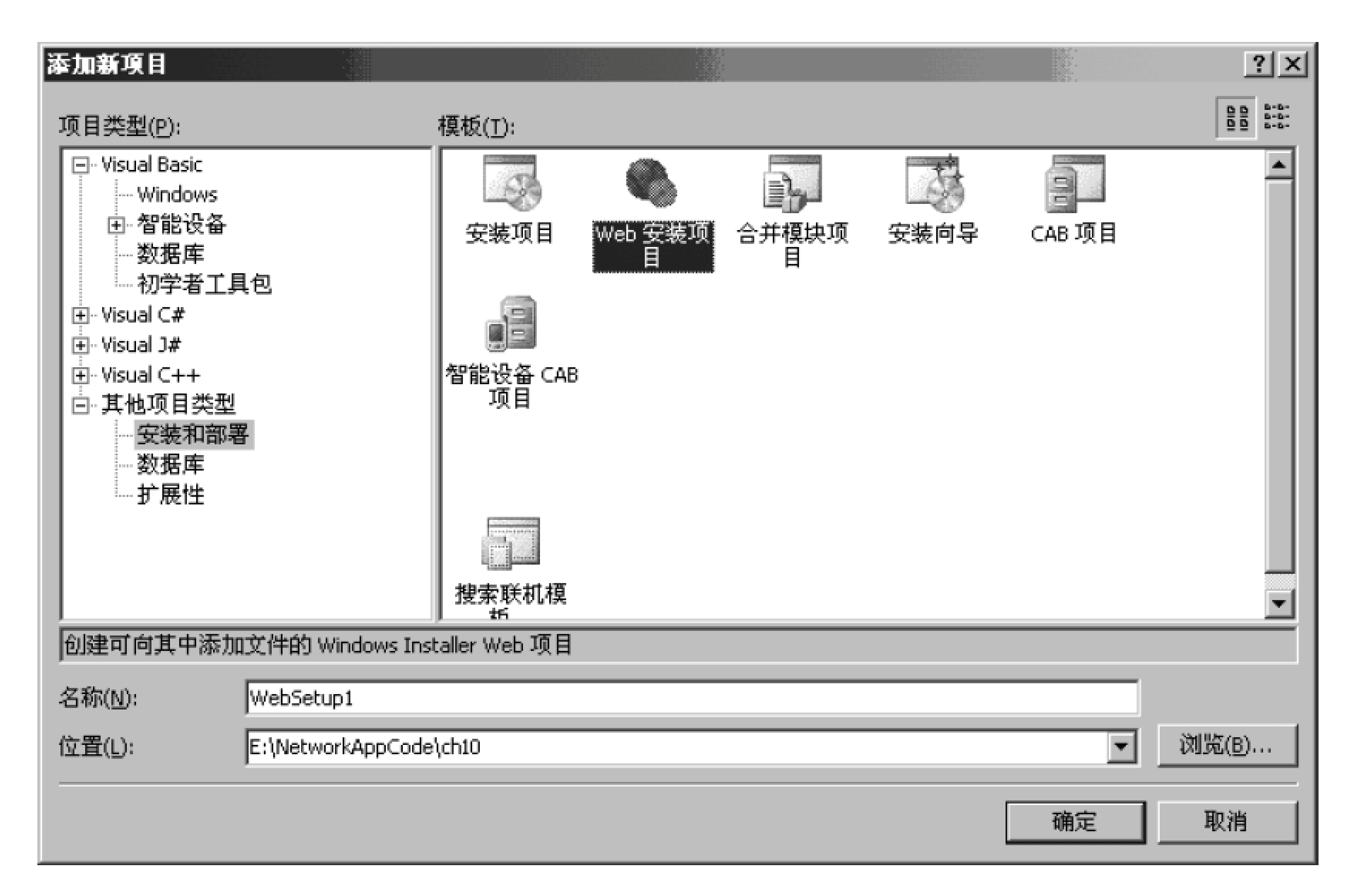


图 10-7　创建 Web 安装项目

单击"确定"按钮，可以看到解决方案中新增了一个安装项目 WebSetup1，并打开了它的"文件系统编辑器"。

将 WebSetup1 的 ProductName 属性改为 WebAppSetup，将 Title 属性改为"安装示例网站"。

在"文件系统编辑器"中选择"Web 应用程序文件夹"节点。

在菜单上选择"操作"→"添加"→"项目输出"。在"添加项目输出组"对话框中，从"项目"下拉列表中选择"WebAppSetup"，如图 10-8 所示。

单击"确定"关闭对话框，在"文件系统编辑器"右侧的文件列表中增加了一行"内容文件"。

在菜单上选择"生成"→"生成 WebSetup1"。生成完成后，到操作系统下找到 WebSetup1 项目目录，打开其下的 Debug 子目录，可以看到两个文件：WebSetup1. msi 和 setup. exe。

将这两个文件复制到 Web 服务器上，执行 setup. exe。

安装程序的执行需要 Microsoft .NET Framework 2.0 环境。如果 Web 服务器已经安装了 Microsoft .NET Framework 2.0，setup. exe 可直接执行；如果未安装，setup. exe 也可自动通过网络下载安装 Microsoft .NET Framework 2.0。

安装程序的执行非常简单。第一步是“欢迎”对话框，第二步是“选择安装地址”对话框，如图 10-9 所示。



图 10-8 添加项目输出组



图 10-9 安装项目的执行

此时可以改变虚拟目录的名称，本例使用 WebAppSetup。继续单击“下一步”按钮，直到安装完成。

执行“计算机管理”工具，可以看到默认 Web 站点下面增加了一个新的虚拟路径 WebAppSetup，如图 10-10 所示。

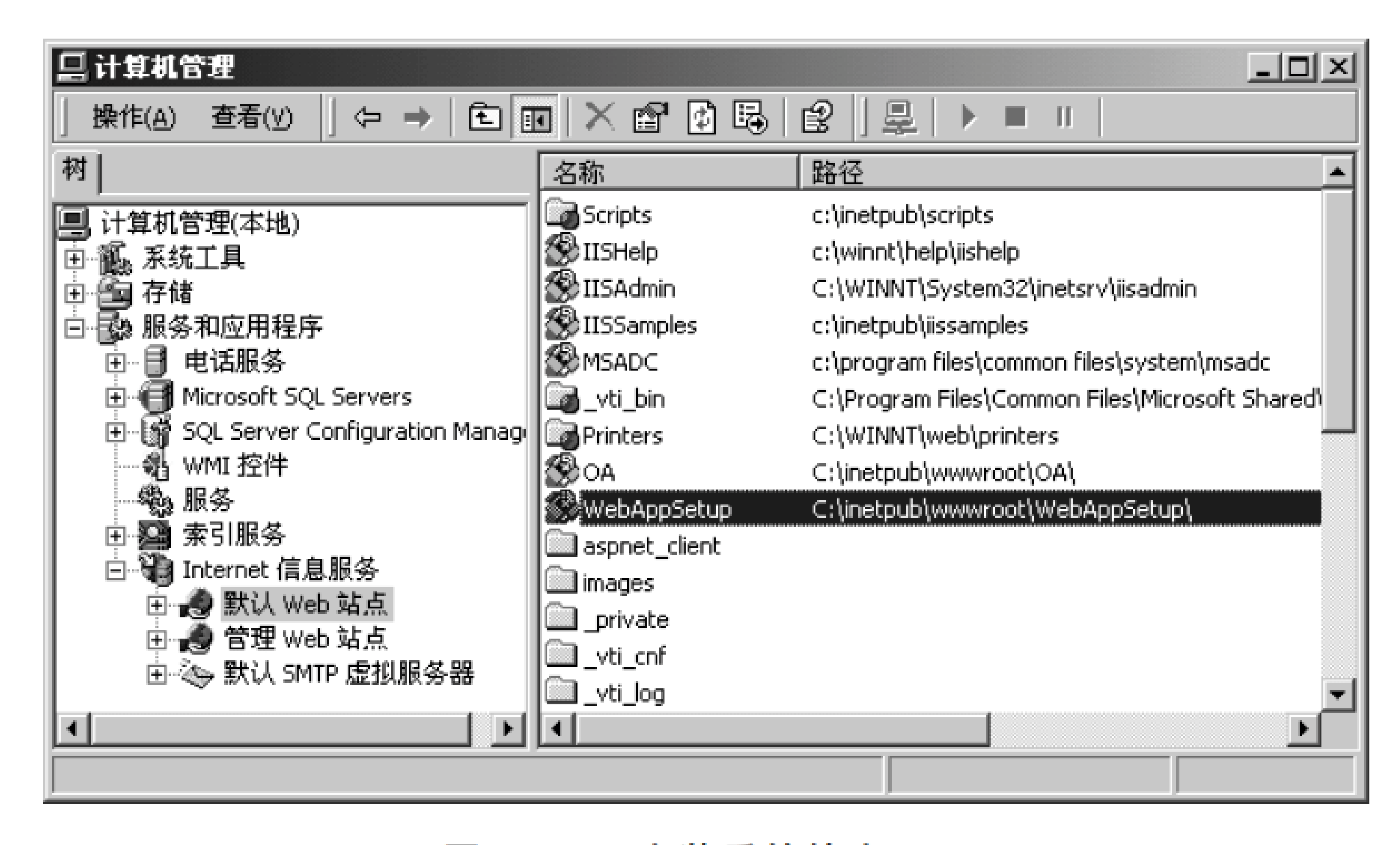


图 10-10 安装后的检查(一)

打开浏览器，在地址栏输入“http://gaoyi/WebAppSetup/”（其中 gaoyi 为机器名），回车后界面如图 10-11 所示，说明安装已经成功。



图 10-11　安装后的检查(二)

习　　题

1. 使用母板页有何意义？

2. 请使用母板页的方法，实现 10.1 节介绍的 UseMasterPage 网站。

3. 导航功能在互联网应用中有什么意义？

4. 请利用 TreeView、ListView 等控件，实现一个类似“资源管理器”的文档管理程序，用于查看一个指定硬盘目录下的文件。

5. 请使用 TreeView 控件，实现 10.2 节介绍的 UseSiteMap 网站。

6. 请自己动手编写 10.3 节介绍的 NewsUC.ascx 和 ActiveOp.ascx 用户控件，并调试运行代码，观察执行效果。

7. 请参考 8.3 节的实例代码，为习题 6 中 NewsUC.ascx 控件隐藏代码文件 NewsUC.ascx.cs 编写 Page_Load()函数，实现从数据库中读取数据，作为数据源绑定到 NewsList 控件的功能。

8. 参照 10.4 节的介绍，采用 Windows Installer 策略，将习题 5 中所创建的网站进行部署。

第 11 章 chapter 11

实用编程技巧

前面各章节介绍的是 Web 应用开发的基本技巧，是每个程序员都需要掌握的。在实际的 Web 应用开发中，根据用户需求的不同，可能需要用到多方面的技巧与知识，所有这些技巧与知识不是任何单独的一本书能够完全涵盖的。作为一个程序员，也不必在开发工作开始之前就务必通晓所有这些知识，完全可以等到需要时，再查找并学习相关的内容；作为一个开发团队，每个成员的分工也会有所不同，也不需要每个人都掌握项目开发的所有细节。

本章介绍 Web 应用开发中的一些实用技巧。在本章中，每一节都可独立成篇，与其他章节的联系并不是太大。每一节的学习都可以丰富读者的开发技能与技巧，但跳过本章也不会影响后面章节的学习。读者也可以在实际开发中根据需要再来查阅本章的相关内容。

11.1 发送电子邮件

电子邮件简称 E-mail，又称为电子信箱。电子邮件在 Internet 的前身 ARPANET 上就已经出现，直到今天，它仍然是 Internet 上应用最广泛的服务之一。

电子邮件是当今人与人之间交互的最常用手段之一。它采用的是异步通信方式，也就是说它不要求通信的双方同时在场(在线)。正是由于它的异步性，使它成为办公自动化等应用的重要基础。

在 Internet 上，电子邮件传输采用最广泛的是简单邮件传输协议(simple mail transfer protocol，SMTP)，接收电子邮件采用的是 POP3(post office protocol 3)协议。

一般用户并不需要了解上述协议的细节，用户可以采用两种方法收发并管理自己的电子邮件：一种是使用专用的邮件客户端，如微软的 Outlook；另一种是登录专门的 Web 邮件网站，在网页上操作。

除了专门对邮件进行收发与管理之外，很多 Web 应用都涉及邮件操作。例如将网站通知发送到用户的邮箱，将用户反馈发送到管理员邮箱，将告警信息发送到工作人员邮箱，将广告信息发送给潜在的客户，利用办公自动化(office automation，OA)系统在同事之间相互发送邮件等。

使用 ASP. NET 所提供的功能发送电子邮件非常简单。在. NET Framework 1. x 中，

使用 System. Web. Mail 命名空间中的类来实现邮件相关功能。. NET Framework 2.0 仍然支持这个命名空间，但仅仅是为了向上兼容。. NET Framework 2.0 宣布该命名空间已经过时，代替它的是 System. Net. Mail 命名空间中的新类。本节就讨论如何使用 System. Net. Mail 命名空间中的类来发送电子邮件。

System. Net. Mail 命名空间中共有 10 多个不同的类，它们都与电子邮件的发送有关。其中两个最核心的类如下。

- MailMessage：电子邮件消息类，它拥有构成一个电子邮件的，如 From、To、Subject 和 Body 等要素属性。
- SmtpClient：该类是一个 SMTP 客户类，通过该类可以把一个邮件(MailMessage 类的一个实例)通过指定的 SMTP 服务器发送到目的地址。

在 . NET Framework 2.0 环境中，无论采用什么语言进行编程，发送电子邮件一般都可以采用如下步骤：

(1) 创建一个 MailMessage 对象。

(2) 将邮件内容各要素赋给 MailMessage 对象的各属性。

(3) 创建一个 SmtpClient 对象。

(4) 指定 SmtpClient 对象所使用的 SMTP 服务器的信息。

(5) 调用 SmtpClient 对象的 Send 方法，发送 MailMessage 对象。

其中第(4)步可以通过设置 SmtpClient 对象的 Host 属性值来完成。但如果已经将 SMTP 服务器信息配置在了 Web. config 文件中，则第(4)步可以省略(本节的实例程序就使用此方法)。

要在 ASP. NET 2.0 页面中发送电子邮件，典型方法是：在邮件页面的“发送”按钮的服务器端单击事件处理函数中完成上述工作。

有关 MailMessage 类和 SmtpClient 类的详细信息本书不再详述，下面仅以一个实例来简单说明在页面中发送邮件的一般方法。

创建一个名为 SendMail 的网站。

为网站增加 Web 配置文件 Web. config。

. NET Framework 2.0 的网络设置架构规定了. NET Framework 连接到 Internet 的方式。根据 . NET Framework 2.0 网络设置架构的规定(本书不讨论其详细内容)，为 Web. config 添加一个<system. net>元素，其代码如下：

```
<system.net>
    <mailSettings>
        <smtp>
            <network host="smtp.****.com" port="25" userName="****"
                password="****"/>
        </smtp>
    </mailSettings>
</system.net>
```

说明：多数程序员习惯将上述代码放在<connectionStrings/>元素的后面，<system.web>元素的前面，但没有硬性规定。其中，<network>元素的 host 属性指定 SMTP 服务器；port 属性指定服务器上 SMTP 服务的端口号，默认为 25；userName 属性用于指定 SMTP 服务器可验证的用户名；password 属性用于指定邮件用户密码。

在默认主页 Default.aspx 上增加一个标题和一个<table>标签，在 table 中输入邮件各要素，并安放一个"发送"按钮，相关代码为：

```
<h2 align="center">发送邮件</h2>
<table style="width: 80%;" align="center" border="0">
<tr>
    <td width="20%">发件人地址:</td>
    <td width="80%">
        <asp:Textbox id="EmailFrom" runat="server" Width="90%"/>
    </td>
</tr>
<tr>
    <td>收件人地址:</td>
    <td>
        <asp:Textbox id="EmailTo" runat="server" Width="90%"/>
    </td>
</tr>
<tr>
    <td>抄送:</td>
    <td>
        <asp:Textbox id="EmailCc" runat="server" Width="90%"/>
    </td>
</tr>
<tr>
    <td>密送:</td>
    <td>
        <asp:Textbox id="EmailBcc" runat="server" Width="90%"/>
    </td>
</tr>
<tr>
    <td>主题:</td>
    <td>
        <asp:Textbox id="EmailSubject" runat="server" Width="90%"/>
    </td>
</tr>
<tr>
    <td>正文:</td>
    <td>
```

```
            <asp:Textbox id="EmailBody" TextMode="MultiLine" Rows="10"
                runat="server" Width="90%"/>
        </td>
    </tr>
    <tr>
        <td>附件:</td>
        <td>
            <asp:FileUpload ID="EmailAttachment" runat="server" Width="90%"/>
        </td>
    </tr>
    <tr>
        <td colspan="2" align="center"><asp:button runat="server" id="SendMail"
            Text="发送" OnClick="SendMail_Click"/></td>
    </tr>
    <tr>
        <td colspan="2" align="center"><asp:Label ID="EmailResult"
            runat="server"></asp:Label></td>
    </tr>
</table>
```

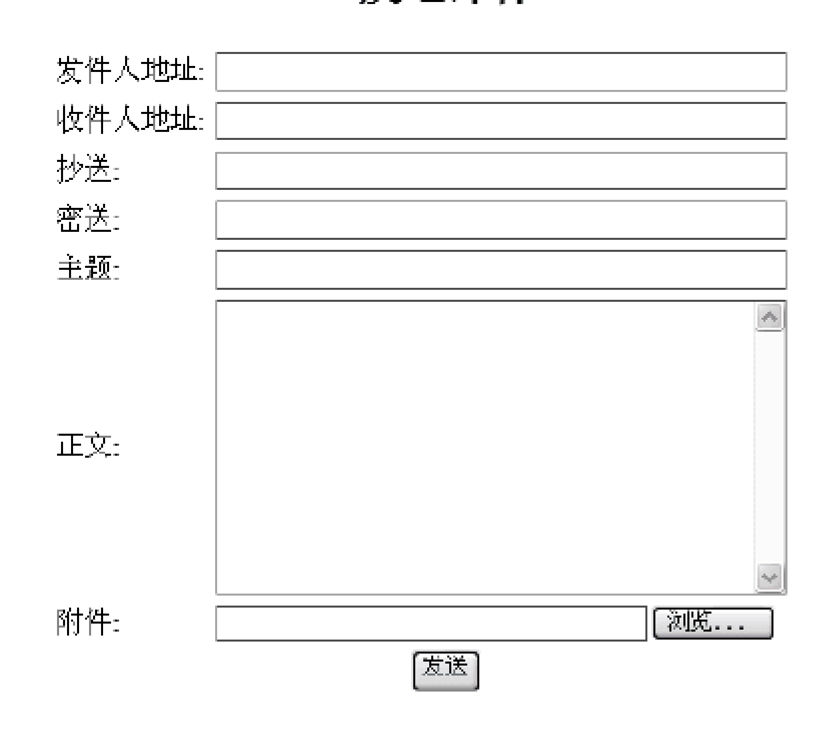


图 11-1　发送邮件页面

在上述代码中，主要是使用 Textbox 控件，供用户输入电子邮件的各要素，如收件人地址、主题和正文等。另外，还使用了一个 FileUpload 控件，用于上传邮件附件，有关 FileUpload 控件的详细介绍参阅第 5 章的 5.2 节；使用了一个按钮，用于提交邮件内容；使用了一个 Label 控件，用于显示邮件的发送结果(可能成功，也可能失败)。

初次执行时，Label 控件上没有内容显示，界面如图 11-1 所示。

为“发送”按钮编写单击事件处理函数，代码如下：

```
protected void SendMail_Click(object sender, EventArgs e)
{
    // (1)创建 MailMessage 对象
    MailMessage mm = new MailMessage(EmailFrom.Text, EmailTo.Text);
    // (2)MailMessage 属性赋值
    if (EmailCc.Text != "")
    {
        MailAddress ma = new MailAddress(EmailCc.Text);
        mm.CC.Add(ma);
    }
    if (EmailBcc.Text != "")
```

```
        {
            MailAddress ma = new MailAddress(EmailBcc.Text);
            mm.Bcc.Add(ma);
        }
        mm.Subject = EmailSubject.Text;
        mm.Body = EmailBody.Text;
        mm.IsBodyHtml = False;
        //填加附件
        mm.Attachments.Add(new
            Attachment(EmailAttachment.PostedFile.InputStream,
            EmailAttachment.FileName));
        // (3)创建 SmtpClient 对象
        SmtpClient sc = new SmtpClient();
        // (4)由于使用了 Web.config 设置,故代码中省略 SMTP 服务器的配置
        // (5)发送 MailMessage
        try
        {
            sc.Send(mm);
            EmailResult.Text = "发送邮件成功。";
        }
        catch (Exception t)
        {
            EmailResult.Text = "发送邮件时发生异常:" + t.Message;
        }
    }
```

说明:上述代码实现了前文所述发送邮件的5个步骤。

发件人地址和收件人地址既可以通过 MailMessage 对象的 From 和 To 属性来设定,也可以在 MailMessage 对象的构造函数中直接指定,如上述代码所示。

MailMessage 对象的 To(收件人)属性、CC(抄送)属性和 Bcc(密送)属性都可以包含多个地址,所以都是地址(MailAddress 对象)集合对象,在上述代码中已经给出了典型的操作方法。

MailMessage 对象的 Attachments 属性是附件集合。每个附件既可以是文件,也可以是流。代码中将上传的文件流直接填加到 Attachments 属性上。

邮件的发送可能会因为各种原因不成功,如用户验证未通过、邮件地址错和 SMTP 服务器错等。当邮件发送出错时可以引发异常,读者在编程时可以有针对性地进行处理。

11.2 使用 Socket 进行通信

20世纪80年代初,美国国防部高级研究计划局(Advanced Research Projects Agency,ARPA)给加利福尼亚大学伯克利分校提供了资金,让他们在 UNIX 操作系统上

实现 TCP/IP 协议。在这个项目中，研究人员为 TCP/IP 网络通信开发了一个 API(应用程序接口)。这个 API 称为 Socket(套接字)接口。今天，Socket 接口是 TCP/IP 网络最为通用的 API，也是在 Internet 上进行通信应用开发最为通用的 API。

20 世纪 90 年代初，由微软公司联合其他几家公司共同制定了一套 Windows 下的网络编程接口，即 Windows Sockets 规范，简称 WinSock。它是 Berkeley Sockets 的重要扩充，主要是增加了一些异步函数，并增加了符合 Windows 消息驱动特性的网络事件异步选择机制。Windows Sockets 规范是一套开放的、支持多种协议的 Windows 下的网络编程接口。从 1991 年的 1.0 版到 1995 年的 2.0.8 版，经过不断完善并在 Intel、Microsoft、Sun、SGI、Informix 和 Novell 等公司的全力支持下，已成为 Windows 网络编程事实上的标准。目前，在实际应用中的 Windows Sockets 规范主要有 1.1 版和 2.0 版，两者的最重要区别是 1.1 版只支持 TCP/IP 协议，而 2.0 版可以支持多协议，2.0 版有良好的向后兼容性。目前，Windows 下的 Internet 软件基本上都是基于 Windows Sockets 开发的。

Socket 实际是在计算机中提供了一个通信端口，应用程序在网络上传输、接收的信息都通过这个 Socket 接口来实现。在应用开发中就像使用文件句柄一样，可以对 Socket 句柄进行读、写操作。使用 Socket 进行通信，在接收端需要等待任意数量的客户连接，以便为它们的通信请求提供服务。对接收端监听的连接来说，它必须在一个已知的名字上。在 TCP/IP 中，这个名字就是本地的 IP 地址，加上一个端口编号。

有关 Socket 通信的详细说明本书从略，仅给出一个实例，说明在 ASP.NET 应用程序中使用 Socket 进行通信的一般方法。

基于 ASP.NET 的应用程序在浏览器中执行，是一个浏览器与应用服务器之间相互通信的过程。但有时也需要在 ASP.NET 应用程序中与远程应用程序进行通信，如在浏览器上监控远程设备的运行情况，必要时向远程设备发送反控命令，发送反控命令时就需要与远程设备中的系统进行通信。这种通信由用户在浏览器上发起，但实际上并不是在浏览器上完成通信，而往往是由 ASP.NET 应用服务器在接到用户的通信请求后，再与远程系统建立基于 Socket 的通信，通信过程如图 11-2 所示。

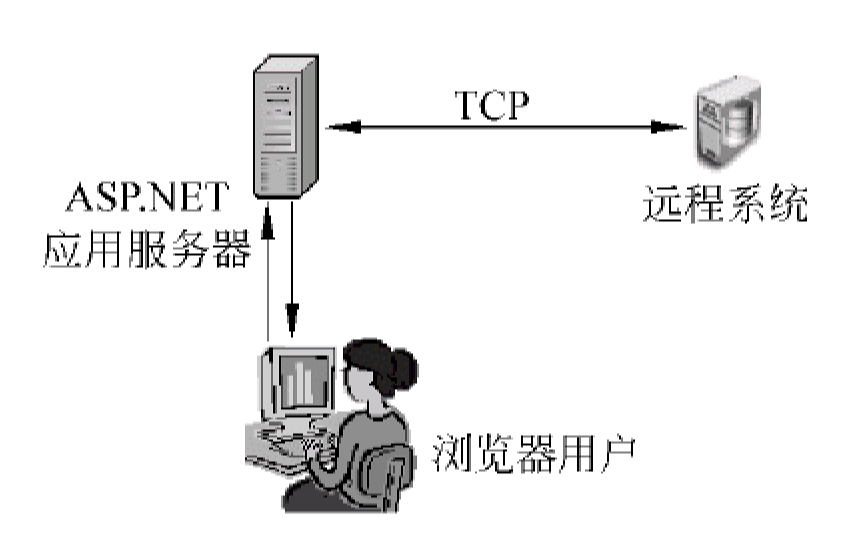


图 11-2 ASP.NET 应用程序中的通信示意图

在上述过程中，ASP.NET 应用服务器一般是根据用户的请求发起通信，需要监听并等待接收的时候很少。但是，为了演示使用 Socket 进行通信的全过程，又避免涉及远程系统的编程(在远程系统中至少应该包括一个通信服务程序)，本节实现一个在网页间进行 Socket 通信的实例(实际的通信仍然是在 ASP.NET 应用服务器上完成的)。

本节实例的功能为：用户同时打开两个页面，分别用于接收和发送。当用户在接收页面上单击“接收”按钮时，向 ASP.NET 应用服务器发送请求，服务器启动监听。当用户在发送页面上输入一段文字，然后单击“发送”按钮时，应用服务器请求向监听端口建立 TCP 连接，并在连接建立后将用户所输入的内容发送给对方。当服务器接收到发送来的内容后，断开连接，并将接收到的内容返回给接收页面。

注意：这不是一个实用的功能，但其实现方法却是实用的。

创建一个名为 TestSocket 的网站。

为网站创建一个名为 Send 的页面。在页面上增加一个 Label 控件、一个 TextBox 控件和一个 Button 控件，代码如下：

```
<asp:Label ID="Label1" runat="server" Text="第 1 次加载发送页面。请输入要发送的内容:"></
asp:Label><br />
<asp:TextBox ID="TextBox1" runat="server" Width="294px"></asp:TextBox>
<br />
<asp:Button ID="Button1" runat="server" OnClick="Button1_Click" Text="发送"/>
```

为"发送"按钮的单击事件处理函数编码如下：

```
protected void Button1_Click(object sender, EventArgs e)
{
    //创建用于发送数据的 Socket
    Socket mySocket = new Socket(AddressFamily.InterNetwork,
        SocketType.Stream, ProtocolType.Tcp);
    //设置目的地址及目的端口号
    IPEndPoint destPoint = new IPEndPoint(IPAddress.Parse("127.0.0.1"),
        8099);

    //将用户输入的文本复制到发送缓冲区
    string str = TextBox1.Text;
    byte[] byteArray = System.Text.Encoding.Default.GetBytes(str);

    //向目的地址请求建立连接
    mySocket.Connect(destPoint);
    //发送数据
    mySocket.Send(byteArray, str.Length, SocketFlags.None);

    //关闭 Socket
    mySocket.Shutdown(SocketShutdown.Send);
    mySocket.Close();

    //向用户页面返回结果
    Label1.Text = "发送成功。";
}
```

从上述代码可以看出，使用 WinSock 进行通信编程，在发送端需要完成如下工作：

(1) 用 Socket 创建一个套接字。

(2) 解析服务器名。

(3) 用 connect 初始化一个连接。

(4) 发送数据。

为网站创建一个名为 Receive 的页面。在页面上增加一个 Label 控件和一个 Button 控件,代码如下:

```
<asp:Label ID="Label1" runat="server" Text="第 1次加载接收页面。">
</asp:Label><br />
<asp:Button ID="Button1" runat="server" Text="接收" OnClick="Button1_Click"/>
```

为“接收”按钮的单击事件处理函数编码如下:

```
protected void Button1_Click(object sender, EventArgs e)
{
    //创建监听 Socket
    Socket mySocket = new Socket(AddressFamily.InterNetwork,
        SocketType.Stream, ProtocolType.Tcp);
    //设置监听地址及端口号
    IPEndPoint destPoint = new IPEndPoint(IPAddress.Parse("127.0.0.1"),
        8099);

    //绑定监听端口
    mySocket.Bind(destPoint);
    //启动监听
    mySocket.Listen(2);

    //创建接收数据缓冲区
    Byte[] recBuffer = new Byte[200];

    //监听到连接请求时,创建接收数据 Socket
    Socket recSocket = mySocket.Accept();

    //接收数据
    int i = recSocket.Receive(recBuffer);
    if (i > 0)
    {
        //将接收到的数据复制到一个字符串
        string str = System.Text.Encoding.Default.GetString(recBuffer);
        //在网页上显示接收到的信息
        Label1.Text = "接收数据的字符数为:" + i.ToString() + "<br>"
            + "接收的数据为:" + str;
    }

    //关闭接收 Socket
    recSocket.Shutdown(SocketShutdown.Receive);
    recSocket.Close();

    //关闭监听 Socket
```

```
    mySocket.Close();
}
```

从上述代码可以看出，使用 WinSock 进行通信编程，在接收端需要完成如下工作：

(1) 将指定协议的套接字绑定到它已知的名字上。这个过程是通过 API 调用 bind 来完成的。

(2) 将套接字置为监听模式。这是用 API 函数 listen 来完成的。

(3) 若一个客户机试图建立连接，服务器必须通过 accept 调用来接受连接。

本实例的执行过程如下：

(1) 执行接收页面，如图 11-3 所示。单击“接收”按钮，应用服务器在服务器端启动端口监听，接收页面暂时不会重新加载。

(2) 另外打开一个浏览器窗口，执行发送页面，如图 11-4 所示。

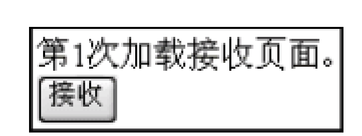


图 11-3　接收页面（第 1 次加载）

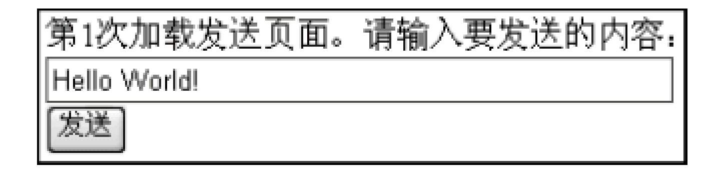


图 11-4　发送页面（第 1 次加载）

(3) 在发送页面的输入框中输入一段文本，然后单击“发送”按钮，发送成功后的界面如图 11-5 所示。

(4) 服务器端的监听进程在接收到文本后，将内容返回给接收页面。此时接收页面重新加载，如图 11-6 所示。

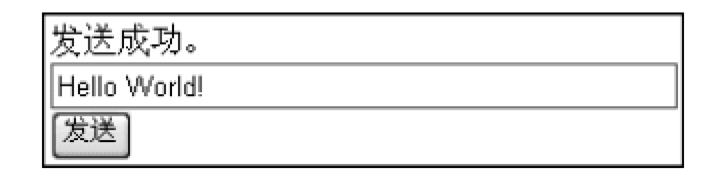


图 11-5　发送页面（发送成功后）

网页在2008-3-5 20:49:17接收的数据大小为：12
接收的数据为：Hello World!
接收

图 11-6　接收页面（接收之后）

11.3　使用 Excel 表格

在 Web 应用中，经常需要以表格的形式展示数据。有多种方法可以实现此功能，如：

(1) 使用 Html Table，在网页上生成表格显示数据。这种方法原理比较简单，我们已经掌握。但这种方法显示效果一般、灵活性较差，对打印功能的支持也较差。

(2) 使用第三方报表工具。这种方法功能强大、灵活，效果最好，但实现方法复杂且与具体的工具有关，这超出了本书范围，这里不做介绍。

(3) 根据显示结果生成 Excel 文件。生成的 Excel 文件可在用户浏览器上直接显示，也可供用户下载后进一步地修改、打印等。在没有使用第三方报表工具的情况下，这不失为一种折中的方法。

本节介绍在 ASP.NET 中使用 Excel 表格的一般方法，首先看一个简单的实例。

创建一个名为 TestExcel 的网站。

创建一个 Excel 文档，文件名为 Test. xls，并将其添加到网站的 App_Data 子目录。文档内容无特别限制。

为网站创建一个名为 ReadExcel 的页面。在页面上增加一个 Label 控件和一个 DataGrid 控件，代码如下：

```
<asp:Label id="Label1" runat="server">从 test.xls 文件中读取数据，显示在下面的 DataGrid控件
            中。</asp:label>
<asp:DataGrid id=DataGrid1 runat="server"/>
```

为页面的 Page_Load()函数编码如下：

```
protected void Page_Load(object sender, EventArgs e)
{
    //指定连接字符串
    string connectionString = "Provider=Microsoft.Jet.OLEDB.4.0;"
        + "Data Source="
        + Server.MapPath(Request.ApplicationPath)
        + "\\App_Data\\Test.xls;Extended Properties=Excel 8.0;";
    //创建一个 DataSet 对象
    DataSet ds = new DataSet();
    OleDbConnection conn = new OleDbConnection(connectionString);
    //创建一个 command 对象
    OleDbCommand command;
    //取文件中的所有数据
    command = new OleDbCommand("Select * From [Sheet1$ ]", conn);
    OleDbDataAdapter da = new OleDbDataAdapter(command);
    try
    {
        //读数据并作为数据源绑定到 DataGrid
        da.Fill(ds);
        DataGrid1.DataSource = ds.Tables[0].DefaultView;
        DataGrid1.DataBind();
    }
    catch
    {
        Label1.Text = "读 Excel 文件时发生异常。";
    }
}
```

执行效果如图 11-7 所示。

从test.xls文件中读取数据，显示在下面的DataGrid控件中。

序号	姓名	性别	出生日期	工资	备注
1	张三	男	1970-2-8 0:00:00	3000	部门经理
2	李四	女	1974-2-9 0:00:00	2600	
3	王五	男	1979-2-10 0:00:00	2400	
4	赵六	女	1970-2-11 0:00:00	2900	系统架构师
5	刘七	男	1988-2-12 0:00:00	2200	

图 11-7 从 Excel 文件中读取数据

说明：在 ASP. NET 中，可使用与操作数据库相似的方法操作 Excel 数据。在 ASP. NET 中操作 Excel 数据时，与操作 Access 数据库相同，同样是通过.NET提供的 OLE DB 托管提供程序进行，

但在细节上有所区别。

1. 建立与 Excel 数据源的连接

与 Excel 数据源建立连接时，常用的连接字符串格式如下：

```
Provider=Microsoft.Jet.OLEDB.4.0;Data Source=目录\文件名.xls;
Extended Properties=Excel 8.0;
```

其中，数据源的路径需要使用物理绝对路径，这一点与使用 Access 数据源相同。不同的是，上述连接字符串中增加了 Extended Properties 关键字，用于设置 Excel 数据源的特有属性。

首先需要设置的是 Excel 的版本属性，具体方法如下：

- 对于 Microsoft Excel 8.0（97）、9.0（2000）、10.0（2002）和 11.0（2003）工作簿，使用 Excel 8.0（如本节实例的连接字符串中）。
- 对于 Microsoft Excel 5.0 和 7.0（95）工作簿，使用 Excel 5.0。
- 对于 Microsoft Excel 4.0 工作簿，使用 Excel 4.0。
- 对于 Microsoft Excel 3.0 工作簿，使用 Excel 3.0。

与一般的数据库表不同，在 Excel 工作簿中，列的名字和数据的值是没有区别的，都放在数据区中。当从 Excel 工作簿中取数据时，默认其第 1 行为标题行；如果此行某单元格为空，则自动生成类似 F1、F2 的名字补充。

如果希望第一行也作为数据显示，还需要设置 Extended Properties 属性的 HDR 参数。如果将 HDR 参数的值设置为 No（默认为 Yes），系统则认为 Excel 工作簿中全部是数据，并自动为各列加上如 F1、F2 形式的名称，其中的数字与单元格的列位置一致，从 1 开始计数。

需要注意的是，当需要为 Extended Properties 属性设置多个参数的值时，必须将多个参数的设置用引号统一括起来（否则会报错），其形式如下：

```
Extended Properties='Excel 8.0;HDR=NO';
```

在显示 Excel 工作簿中的数据时，还可能会遇到这样的情况，就是有些有效单元格数据显示不出来。出现这种情况的原因可能是：系统根据前面单元格推断后续单元格的数据类型，所以当认为后续单元格数据类型不符时，就不显示其中的数据。可以通过将 Extended Properties 属性的 IMEX 参数设置为 1，这样就会将 Excel 工作簿中类型不统一的列都作为文本读取，如：

```
Extended Properties='Excel 8.0;HDR=NO;IMEX=1;';
```

2. 操作 Excel 数据

与操作数据库中的数据相似，也可以用 Select、Insert 等命令语句来操作 Excel 工作簿中的数据。

使用 Select 语句可以查询 Excel 工作簿中的数据。例如，执行语句

```
Select * From [Sheet1$ ]
```

即可取得工作簿 Sheet1 中的所有数据。

与一般的 SQL 语句相比，这个语句有两点不同：一是工作簿名必须用方括号括起来；二是工作簿名后面一定要加一个 $ 符号。

使用 Select 语句同样可以限制查询数据的范围。可以像一般 SQL 语句那样使用 Where 子句：

```
Select * From [Sheet1$ ] where 姓名='张三'
```

还可以按照 Excel 的方式，直接指定范围，如：

```
Select * From [Sheet1$  B2:D4]
```

还可以选择指定列的数据，如：

```
Select 姓名 From [Sheet1]
Select 男 From [Sheet1$  B2:D4]
```

除查询之外，还可以对 Excel 工作簿中的数据进行修改。

可以用 Insert 语句向 Excel 工作簿插入数据，如：

```
Insert into [Sheet1$ ] (姓名,性别) Values ('陈八', '男')
```

可以用 Update 语句对 Excel 工作簿中现有的数据进行修改，如：

```
Update [Sheet1$ ] Set 备注='秘书' where 姓名='李四'
```

可以用 Create 语句创建新的 Excel 工作簿，如：

```
Create Table MySheet  (部门编号 char(255),部门名称 char(255))
```

前面主要介绍了对一个已存在的 Excel 文档中的数据进行操作的方法。在 ASP.NET 中还可以创建一个新的 Excel 文档。

为 TestExcel 网站创建一个名为 WriteExcel 的页面。在页面上增加一个 GridView 控件、一段文字和一个 Button 控件，代码如下：

```
<asp:GridView ID="GridView1" runat="server" AutoGenerateColumns="False"
    BackColor="#C0C0FF" BorderColor="#0000C0" BorderWidth="2px">
  <Columns>
    <asp:BoundField HeaderText="序号" DataField="id"/>
    <asp:BoundField HeaderText="姓名" DataField="Name"/>
    <asp:BoundField HeaderText="性别" DataField="Sex"/>
  </Columns>
</asp:GridView>
GridView控件的定制属性将会原样导出到Excel文件中。<br />
<asp:Button ID="Button1" runat="server" Text="导出" OnClick="Button1_Click"/>
```

注意：代码中对 GridView 控件的外观进行了一些定制，这些外观效果将会原样导

出到 Excel 文件中。

为页面的 Page_Load()函数编码如下：

```
if (! IsPostBack)
{
    //创建并绑定数据源
    GridView1.DataSource = CreateDataTable();
    GridView1.DataBind();
}
```

其中，CreateDataTable()函数动态创建一个 DataTable 对象，生成数据，并将其返回作为 GridView 控件的数据源。CreateDataTable()函数的实现代码如下：

```
DataTable CreateDataTable()
{
    //创建一个 DataTable
    DataTable myDataTable = new DataTable();
    //为 DataTable 创建三个列
    myDataTable.Columns.Add(new DataColumn("id", typeof(Int32)));
    myDataTable.Columns.Add(new DataColumn("Name", typeof(string)));
    myDataTable.Columns.Add(new DataColumn("Sex", typeof(string)));
    //为 DataTable 增加数据
    for (int i = 0; i < 6; i++)
    {
        //创建一个 DataRow
        DataRow myDataRow;
        myDataRow = myDataTable.NewRow();
        //创建 DataRow 的数据
        myDataRow[0] = i;
        myDataRow[1] = "张" + i.ToString();
        if ((i % 2)==0)
            myDataRow[2] = "男";
        else
            myDataRow[2] = "女";
        //将 DataRow 加入到 DataTable 中
        myDataTable.Rows.Add(myDataRow);
    }
    return myDataTable;
}
```

为页面的“导出”按钮实现单击事件处理函数如下：

```
protected void Button1_Click(object sender, System.EventArgs e)
{
    //设置 Response 对象属性
    Response.Clear();
```

```
        Response.Buffer = True;
        Response.Charset = "GB2312";
        Response.AppendHeader("Content-Disposition",
            "attachment;filename=FileName.xls");//以附件的形式返回给客户端
        Response.ContentEncoding = System.Text.Encoding.UTF7;
        Response.ContentType = "application/ms-excel";//设置输出文件类型为 excel
        //将 GridView1 转化为字符串并写入 Response 缓冲区
        StringWriter myStringWriter = new StringWriter();
        System.Web.UI.HtmlTextWriter myHtmlTextWriter =
            new System.Web.UI.HtmlTextWriter(myStringWriter);
        this.GridView1.RenderControl(myHtmlTextWriter);
        Response.Output.Write(myStringWriter.ToString());
        //将缓冲区内容发回客户端
        Response.Flush();
        Response.End();
    }
```

代码中有较详细的注释,其主要功能是直接向客户端输出一个 Excel 文件。当然,适当地设置 Response 对象的 content-disposition 和 ContentType 属性,也可以将内容输出为其他类型的文件。

需要注意的是,StringWriter 在命名空间 System. IO 中定义,因此,在代码文件的开始处必须增加对 System. IO 命名空间的引用。HtmlTextWriter 在命名空间 System. Web. UI 中定义,该命名空间是页面的默认引用,不需要再增加。

执行页面,界面效果如图 11-8 所示。

单击"导出"按钮,系统还不能正常运行,会报一个 HttpException 错。这是因为 GridView 类型的控件要想直接以文件的形式返回给客户端,必须重载页面的 VerifyRenderingInServerForm 方法。所以,只要为页面再增加如下代码就可以了。

```
public override void VerifyRenderingInServerForm(Control control)
{
}
```

再次执行页面,弹出如图 11-9 所示对话框。

序号	姓名	性别
0	张0	男
1	张1	女
2	张2	男
3	张3	女
4	张4	男
5	张5	女

GridView控件的定制属性将会原样导出到Excel文件中。

导出

图 11-8 自动生成的 GridView 数据



图 11-9 导出到 Excel 文件

将文件直接打开或保存到本地后再打开，可以看到 Excel 文档的内容与 GridView 控件中显示的内容完全相同。

11.4 处理数据库中的图片

处理图片是网站建设的重要内容。

在 ASP. NET 应用程序中，图片可以用文件的形式存放在服务器上，直接用 ASP. NET 控件进行显示，相关内容参阅 4.5 节。另外，图片还可以存放到数据库中，需要时从数据库中取出进行显示。

处理数据库中图片的方式与处理数据库中一般数据(如数字、文本等)的方法不同，主要是不能直接用一般的 SQL 语句进行处理，处理图片的过程要复杂一些。

涉及数据库的内容不是本书重点，本节仅以一个实例为基础，详细介绍数据库中图片数据的处理方式，与数据库有关的部分不做过多的解释。

首先，在本书的示例数据库 NetSchool 中创建一个测试表，用于存储图片。建表语句如下：

```
USE NetSchool
CREATE TABLE [Test_Picture] (
[ID] [int] IDENTITY (1, 1) NOT NULL PRIMARY KEY,
[Pic_Desc] [varchar] (50) NULL,
[Pic_Type] [varchar] (50) NULL,
[Pic_Data] [image] NULL
)
GO
```

创建一个名为 PictureInDB 的网站。

为网站创建 Web 配置文件 Web. config，并增加连接字符串 SQLConnectionString。

为网站创建 ASP. NET 保留文件夹 App_Code，在 App_Code 中创建一个类，名为 Picture. cs。为该类增加一个私有成员 SQLConnectionString。

```
private string SQLConnectionString =ConfigurationManager
.ConnectionStrings["SQLConnectionString"].ConnectionString;
```

由于在该类中需要操作 SQL Server 数据库，因此需要加上对 System. Data. SqlClient 命名空间的引用。

为网站的默认主页增加一个 GridView 控件、一个 TextBox 控件、一个 FileUpload 控件、一个 Button 控件、一个 Label 控件和一个 Panel 控件，Panel 控件上包含一个 Image 控件，代码如下：

```
< asp:GridView ID="GridView1" runat="server" AutoGenerateColumns="False" DataKeyNames="ID"
BackColor="#E0E0E0" BorderColor="#0000C0" BorderWidth="2px" Caption="图片列表" Width="622px" >
    <Columns>
```

```
            <asp:BoundField DataField="ID" HeaderText="ID" InsertVisible="False"
                ReadOnly="True" SortExpression="ID"/>
            <asp:BoundField DataField="Pic_Desc" HeaderText="图片说明"
                SortExpression="Pic_Desc"/>
            <asp:BoundField DataField="Pic_Type" HeaderText="文件类型"
                SortExpression="Pic_Type"/>
            <asp:CommandField ButtonType="Button" SelectText="显示"
                ShowSelectButton="True"/>
        </Columns>
        <SelectedRowStyle BackColor="Gray"/>
    </asp:GridView>
    <h2><span style="color: red">上传新图片</span></h2>
    文件说明:<asp:TextBox ID="Desc" runat="server"></asp:TextBox><br />
    文 件 名:<asp:FileUpload ID="FileUpload1" runat="server"/><br />
    <asp:Button ID="Button1" runat="server" Text="图片上传"/><br />
    <asp:Label ID="Label1" runat="server"></asp:Label><br />
    <asp:Panel ID="Panel1" runat="server" Height="50px" Width="125px">
        <asp:Image ID="Image1" runat="server"/>
    </asp:Panel>
```

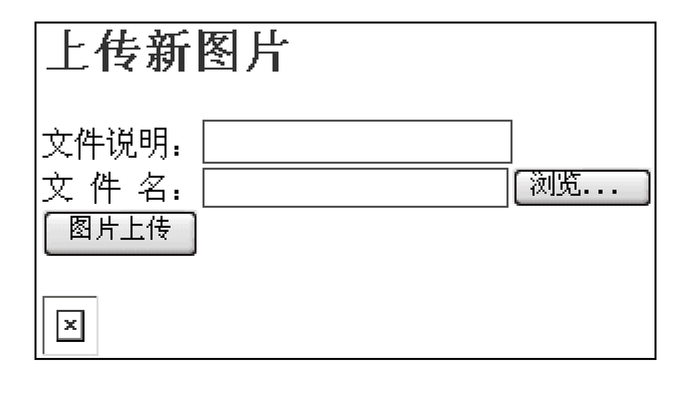


图 11-10 页面的原始效果

页面此时可以执行,执行效果如图 11-10 所示。

可以看到:由于没有为 GridView 指定数据源,所以 GridView 控件不显示。由于没有为 Image 控件指定 ImageUrl 属性,所以该控件在页面上显示为一个叉形图标。

为 Picture 类增加一个成员函数,代码如下:

```
public SqlDataReader GetAllPictures()
{
    SqlConnection myConnection = new SqlConnection(SQLConnectionString);
    string sql = "SELECT ID,Pic_Desc,Pic_Type FROM Test_Picture ORDER BY ID";
    SqlCommand myCommand = new SqlCommand(sql,myConnection);
    myCommand.CommandType = CommandType.Text;

    SqlDataReader myDataReader = null;

    try
    {
        //连接数据库
        myConnection.Open();
    }
    catch (Exception ex)
    {
        throw new Exception("连接数据库失败!", ex);
```

```
    }

    try
    {
        //取所有的图片信息
        myDataReader =
            myCommand.ExecuteReader(CommandBehavior.CloseConnection);
    }
    catch (Exception ex)
    {
        throw new Exception(ex.Message, ex);
    }

    return myDataReader;
}
```

该函数的功能是从数据库表中取所有的图片信息，并将结果集以一个 SqlDataReader 对象的形式返回。

为默认主页的代码页 Default.aspx.cs 增加一个函数，代码如下：

```
private void ListPictures()
{
    //取所有图片信息
    Picture picTrue = new Picture();
    SqlDataReader myDataReader = picTrue.GetAllPictures();

    //设置 GridView 控件的数据源，并绑定数据
    GridView1.DataSource = myDataReader;
    GridView1.DataBind();

    //关闭数据源
    myDataReader.Close();
}
```

由于使用了 SqlDataReader 对象，因此需要加上对 System.Data.SqlClient 命名空间的引用。

修改页面的 Page_Load()函数，代码如下：

```
protected void Page_Load(object sender, EventArgs e)
{
    if (!Page.IsPostBack)
    {
        //第 1 次加载页面时将 Panel 控件设为不可见
        this.Panel1.Visible = False;
        //取所有图片信息并列表
```

```
            ListPictures();
        }
    }
```

再次执行页面，由于数据库表中还没有数据，所以 GridView 控件仍然不显示。但由于 Panel1 的 Visible 属性被设为了 False，所以叉形图标不再显示了。

再为 Picture 类增加一个成员函数，代码如下：

```
public void AddPicture(String Pic_Desc, byte[] Pic_Data, String Pic_Type)
{
    SqlConnection myConnection = new SqlConnection(SQLConnectionString);
    string sql = "insert Test_Picture (Pic_Desc,Pic_Type,Pic_Data)
        values (@ Pic_Desc,@ Pic_Type,@ Pic_Data)";// 插入一个图片
    SqlCommand myCommand = new SqlCommand(sql, myConnection);
    myCommand.CommandType = CommandType.Text;

    // 创建访问数据库的参数
    SqlParameter paraPic_Desc = new SqlParameter("@ Pic_Desc",
        SqlDbType.VarChar, 50);
    paraPic_Desc.Value = Pic_Desc;
    myCommand.Parameters.Add(paraPic_Desc);

    SqlParameter paraPic_Type = new SqlParameter("@ Pic_Type",
        SqlDbType.VarChar, 50);
    paraPic_Type.Value = Pic_Type;
    myCommand.Parameters.Add(paraPic_Type);

    SqlParameter paraPic_Data = new SqlParameter("@ Pic_Data",
        SqlDbType.Image);
    paraPic_Data.Value = Pic_Data;
    myCommand.Parameters.Add(paraPic_Data);

    try
    {
        // 连接数据库
        myConnection.Open();
    }
    catch (Exception ex)
    {
        throw new Exception("连接数据库失败！", ex);
    }

    try
    {
        // 插入一个图片
```

```
            myCommand.ExecuteNonQuery();
        }
        catch (Exception ex)
        {
            throw new Exception(ex.Message, ex);
        }
        finally
        {
            if (myConnection.State == ConnectionState.Open)
            {
                //关闭数据库连接
                myConnection.Close();
            }
        }
    }
```

该函数的功能是向数据库表插入一个包含图片字段的记录。请注意在 SqlCommand 对象中使用 SQL 语句和设置参数值的方法,这是向数据库表插入图片数据的一般方法。也可以使用数据库的存储过程完成上述功能。

为默认主页的“图片上传”按钮实现单击事件处理函数,代码如下:

```
protected void Button1_Click(object sender, EventArgs e)
{
    string str = "";
    //如果 FileUpload控件包含文件
    if (FileUpload1.HasFile)
    {
        Picture picture = new Picture();
        //定义图片的 IO流
        Stream myStream = FileUpload1.PostedFile.InputStream;
        //创建字节数组
        byte[] pictureData = new byte[FileUpload1.PostedFile.ContentLength];
        try
        {
            //将图片数据保存到字节数组中
            myStream.Read(pictureData, 0,
                FileUpload1.PostedFile.ContentLength);
            //将上传的图片保存到数据库
            picture.AddPicture(Desc.Text.Trim(), pictureData,
                FileUpload1.PostedFile.ContentType);
        }
        catch (Exception ex)
        {
            //如果文件保存时发生异常,则显示异常信息
```

```
            str += "保存文件出错:" + ex.Message;
        }
        //上传并保存图片成功
        Desc.Text = "";
        str = "上传图片成功。";
        ListPictures();
    }
    else
    {
        //如果不包含文件,给出提示
        str = "无上传文件。";
    }
    Label1.Text = str;
}
```

上述代码的功能为:将上传的文件先暂存在一个字节数组中,然后向数据库插入一条数据。

由于使用了 Stream 对象,因此需要加上对 System.IO 命名空间的引用。

再次执行页面,输入文件说明,并选择适当的图片上传。页面重新加载后,图片列表出现,并显示"上传图片成功"提示,如图 11-11 所示。

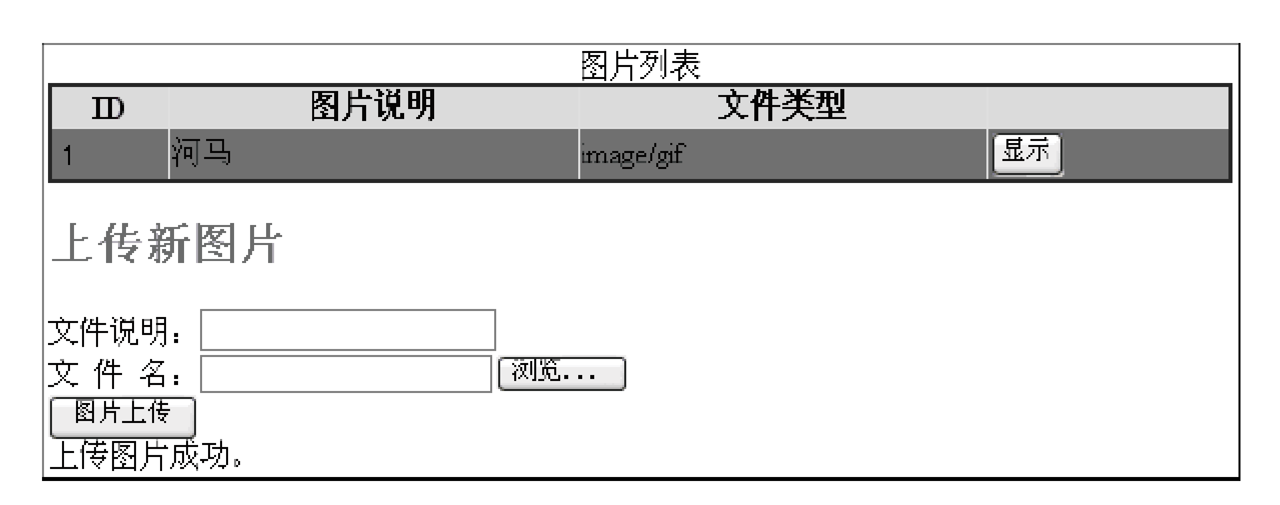


图 11-11 上传图片后页面的效果

此时,数据库中的图片还不能显示。

再为 Picture 类增加一个成员函数,代码如下:

```
public SqlDataReader GetSinglePicture(int ID)
{
    SqlConnection myConnection = new SqlConnection(SQLConnectionString);
    SqlCommand myCommand = new SqlCommand(
    "Select * from Test_Picture where ID=" + ID.ToString(), myConnection);
    myCommand.CommandType = CommandType.Text;

    SqlDataReader myDataReader = null;

    try
    {
        //连接数据库
```

```
            myConnection.Open();
        }
        catch (Exception ex)
        {
            throw new Exception("连接数据库失败!", ex);
        }

        try
        {
            //从数据库中取一条图片记录
            myDataReader =
                myCommand.ExecuteReader(CommandBehavior.CloseConnection);
        }
        catch (Exception ex)
        {
            throw new Exception(ex.Message, ex);
        }

        return myDataReader;
    }
```

该函数的功能是根据图片 ID 从数据库中取一条记录。

为 GridView1 的 OnSelectedIndexChanged 事件实现处理函数,代码如下:

```
protected void GridView1_SelectedIndexChanged(object sender, EventArgs e)
{
    //置 Panel1 为可见
    this.Panel1.Visible = True;
    //取图片 ID
    GridViewRow row = GridView1.SelectedRow;
    String ID = row.Cells[0].Text;
    //指定图片的 ImageUrl 属性
    Image1.ImageUrl = "~/PictureDisp.aspx?ID=" + ID;
}
```

该函数的功能是:首先将 Panel1 的 Visible 属性设为 True,使其在页面上可见。然后从页面上的 GridView 控件中取所选图片的 ID。最后修改 Image1 的 ImageUrl 属性。

在代码中将 Image1 的 ImageUrl 属性指向了一个页面 PictureDisp.aspx(一般是指向一个图片文件),并将所选图片的 ID 作为参数传递给该页面。可以想象,将在该页面中取所选图片的数据并返回。

为网站新创建一个页面 PictureDisp.aspx。该页面不必有界面,所以直接打开其隐藏代码页,修改其 Page_Load()函数,代码如下:

```
protected void Page_Load(object sender, EventArgs e)
```

```
{
    if (Request.Params["ID"] != null)//如果图片 ID不为空
    {
        Picture picture = new Picture();
        //从数据库中取图片数据,到 SqlDataReader
        SqlDataReader myDataReader =
picture.GetSinglePicture(Int32.Parse(Request.Params["ID"].ToString()));
        //创建字节数组,用来保存图片数据
        byte[] pictureData = null;
        //从 SqlDataReader 中取图片数据,到字节数组
        while (myDataReader.Read())
        {
            pictureData = (byte[])myDataReader["Pic_Data"];
            Response.ContentType = myDataReader["Pic_Type"].ToString();
        }
        myDataReader.Close();

        //将图片数据返回到客户端
        Response.AppendHeader("Content-Length",
            pictureData.Length.ToString());
        Response.BinaryWrite(pictureData);
        Response.End();
    }
}
```

同样,由于使用了 SqlDataReader 对象,因此需要加上对 System.Data.SqlClient 命名空间的引用。

该函数的功能是:从数据库中取出图片数据,直接经由 Response 对象返回给客户端;然后调用 Response 对象的 End()函数,页面本身的内容就不会再向客户端发送了。

因此,此页面的调用效果是仅返回图片数据。由此也可以看出,将 Image1 的 ImageUrl 属性指向此页面,与将 ImageUrl 属性指向一个图片文件,在客户端的效果是一样的。

再次执行默认主页,可以多上传几个图片,单击每个图片后面的“显示”按钮,页面的下部就会显示该图片。

11.5 在程序中操作图片

在 ASP.NET 应用程序中不仅可以显示已有的图片,还可以在程序中生成新图片或对已有图片进行修改。

System.Drawing 命名空间提供了对 GDI+基本图形功能的访问。Graphics 类在 System.Drawing 命名中定义,提供了在程序中绘制图形的方法。可以使用 Pen 类在 Graphics 上绘制 Rectangle、Line、Ellipse 和 Point 等基本图元;还可以使用从抽象类

Brush 派生出的类填充形状的内部、在图片上写文本等。

创建一个名为 ImageProgram 的网站。

为网站创建一个名为 CreateImage.aspx 的页面，该页面本身没有界面，而是创建一个图片，并将该图片直接返回到客户端。直接打开该页面的隐藏代码文件，增加对图形操作命名空间的引用，代码如下：

```
using System.Drawing;
using System.Drawing.Imaging;
```

修改 Page_Load()函数的实现，代码如下：

```
protected void Page_Load(object sender, EventArgs e)
{
    //创建一个空图片
    Bitmap mymap = new Bitmap(200, 300);
    Graphics graphic = Graphics.FromImage(mymap);

    //绘制椭圆
    graphic.DrawEllipse(new Pen(Color.Red, 3), 10, 10, 50, 80);
    //绘制矩形
    graphic.DrawRectangle(new Pen(Color.Green, 4), 10, 100, 60, 100);
    //绘制线
    graphic.DrawLine(new Pen(Color.Blue, 6), 10, 250, 80, 250);
    //释放占用的资源
    graphic.Dispose();

    //将新图片返回到客户端
    mymap.Save(Response.OutputStream, ImageFormat.Jpeg);

    mymap.Dispose();
}
```

上述代码的功能是：首先创建一个宽 200 像素、高 300 像素的图片，并用该图片创建一个 Graphics 对象实例 graphic。然后在 graphic 上绘制各种形状。最后将经过绘制的图片通过 Response 对象直接返回到客户端。

执行页面，可以看到浏览器上显示一个黑色背景的图片，图片上有一个红色的椭圆、一个绿色的矩形和一个蓝色的线段。

读者在平时上网时可能也会注意到，有些商业网站上的图片都带有该网站的标记（水印），这既是一种宣传措施，也是一种保护措施。使用 ASP.NET 的图片处理功能，上述水印效果很容易实现。

再为网站创建一个名为 DrawStringOnImage.aspx 的页面，同样直接打开其隐藏代码文件，增加对图形操作命名空间的引用，并修改 Page_Load()函数的实现，代码如下：

```
protected void Page_Load(object sender, EventArgs e)
```

```
{
    //取图片文件
    String fileName = Server.MapPath(Request.ApplicationPath)
        + "\\Images\\青海湖.jpg";
    System.Drawing.Image myImage = System.Drawing.Image.FromFile(fileName);

    //创建新图片
    Graphics graphic = Graphics.FromImage(myImage);
    graphic.DrawImage(myImage, 0, 0, myImage.Width, myImage.Height);

    //定义字体和画笔
    Font font = new Font("黑体", 12);
    Brush brush = new SolidBrush(Color.Blue);

    //在新图片上写文字
    graphic.DrawString("美丽的青海湖", font, brush, 10, 10);

    //将新图片返回到客户端
    myImage.Save(Response.OutputStream,
        System.Drawing.Imaging.ImageFormat.Jpeg);

    //是否绘画资源
    graphic.Dispose();
    myImage.Dispose();
}
```

上述代码的功能是：首先将一个已有图片文件加载到一个Image对象中，并用该对象创建一个Graphics对象实例graphic。然后调用graphic的DrawString方法，在图片上绘制文字。最后将经过绘制的图片通过Response对象直接返回到客户端。



图11-12　增加了文字后的图片

执行页面，效果如图11-12所示。

注意：执行此页面之前需要为网站创建合适的子目录并导入一个已有的图片文件，可能还需要根据图片文件名称对上述代码做适当修改。

另外，在System. Drawing. Drawing2D、System. Drawing. Imaging及System. Drawing. Text命名空间中还提供了一些更高级的图形功能，有兴趣的读者可以自行学习。

习　　题

1. 试述电子邮件应用的重要性。Internet上的电子邮件传输一般采用什么协议？

2. .NET Framework 2.0 使用哪个命名空间中的类来实现邮件相关的功能？其中与电子邮件发送有关的核心类有哪些？

3. 简述在.NET Framework 2.0 环境中发送电子邮件编程的一般步骤。

4. 简述什么是 Socket，什么是 WinSock。

5. 试述在 ASP.NET 应用程序中进行 Socket 通信的过程。

6. 使用 WinSock 进行通信编程，在接收端需要做哪些工作？

7. 在 Web 应用中，有哪些方法可以实现以表格的形式展示数据？各有何利弊？

8. 概述在 ASP.NET 中，操作 Excel 文件中数据的方法。

9. 在设置连接字符串时，使用 Excel 数据源与使用 Access 数据源有何异同？

10. 实现 11.3 节的实例，在实例中增加对 HDR 和 IMEX 两个参数的设置，观察程序的执行效果。

11. 试述向数据库表插入一个包含图片字段的记录时，在 SqlCommand 对象中使用 SQL 语句和设置参数值的方法。

12. 试述 11.4 节中 PictureDisp.aspx 页面的功能及实现方法。

13. 在 ASP.NET 应用程序中生成新图片或对已有图片进行修改时，可以使用哪个命名空间中的哪些类？

第 12 章 chapter 12

高级数据库技术

在实际的编程中，还可能涉及一些高层次的数据库操纵技术。了解这些技术，有利于从更高的层次上完成系统设计，实现更复杂的业务逻辑。这些技术一般应用于大型商业网站，本书的应用实例中并没有涉及。

121　使用数据库连接池

前文已经介绍过，为了使网站能够提供高级的服务功能，绝大多数 Web 应用程序都具有对数据库中的大量业务数据进行动态管理的能力。因此，对于一个专业网站，其用户体验与其数据库的操作性能有很大的关系。

要提高数据库的操作性能，除了采用更好的数据库服务器，进行更合理的数据库设计之外，在 Web 应用程序的实现环节也有很多措施可以采取。其中，使用数据库连接池就可以显著提高应用程序的性能和可缩放性。

首先，让我们看一下从应用服务器连接数据库的方式。

1. 即时连接

在程序中操作数据库总是遵循“连接数据库-操作-关闭数据库连接”的过程。当人们最初开始开发基于数据库的所谓“动态 Web 应用”(这是当时很常用的一个词)时，一般都是采用即时连接的方式(见图 12-1)，就是在程序中每次需要对数据库进行操作时都即时连接数据库，操作完成后再关闭连接。

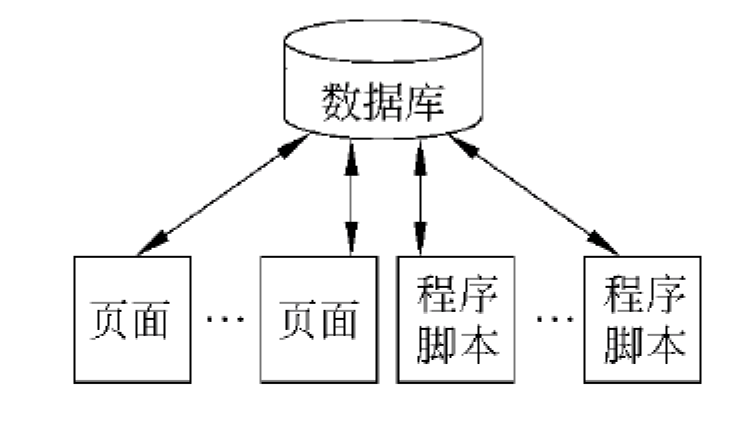


图 12-1　操作数据库的即时连接方式

使用这种方式，每次执行需要访问数据库的页面或程序脚本时都会建立与数据库的连接。这种方法逻辑简单，对于用户量、访问量都很小，应用中很少访问数据库是没有问题的。

但是，在每次连接数据库的过程中，在数据库服务器端都要完成几个耗时的步骤，如：必须建立物理通道(例如套接字或命名管道)，必须与服务器进行初次握手，必须分析连接字符串信息，必须进行身份验证，必须运行事务登记等。

如果用户量大、数据库访问频繁，采用此方法则会极大地降低程序响应速度，过多的

数据库连接也会使数据库服务器不堪负荷。

2. 固定连接

为了克服即时连接方式的缺点，最早解决上述问题的方法是采用固定连接方式(见图 12-2)。

采用固定连接方式，Web 应用程序在服务器端维护一个或几个固定的数据库连接，例如，将这些数据库连接以应用程序变量的形式存储在服务器端并对所有用户可见。对于所有的用户和所有的功能，当需要进行数据库操作时都通过这些连接完成。

采用固定连接方式，不是等到用户请求时再连接数据库，而是当应用程序启动时就连接数据库，当应用程序结束时再断开连接。这样，在处理每个用户的数据库操作请求之前和之后，就不再需要执行相应的连接和断开操作了。这样，也就在一定程度上提高了数据库的操作效率。

但是，采用固定连接方式，当多个用户同时请求数据库操作时，这些用户就会共享同一个连接访问数据库。当用户数量增大后，数据库操作效率很难满足用户的需求。另外，在应用程序运行期间，如果因为意外原因造成应用服务器与数据库连接的断开(如数据库服务器关机或网络不通)，则需要自行编程维护上述固定连接，这是很难实现的。

可见，上述两种方式各有利弊，都很难满足应用程序的性能和伸缩性要求。幸运的是，在当前很多的 Web 应用程序开发与运行环境中，上述问题都得到了很好的解决。在 ASP. NET 环境下，其解决方法就是采用数据库连接池。

可以说，与其他的 Web 应用程序开发与运行环境相比，在 ASP. NET 环境下这个问题解决得更好。在 ASP. NET 环境下，若原有程序是采用上述即时连接方式实现的(事实上，大多数程序都是这样实现的)，如果要使用数据库连接池，原有程序几乎不需要做任何改动。

使用数据库连接池的原理如图 12-3 所示。

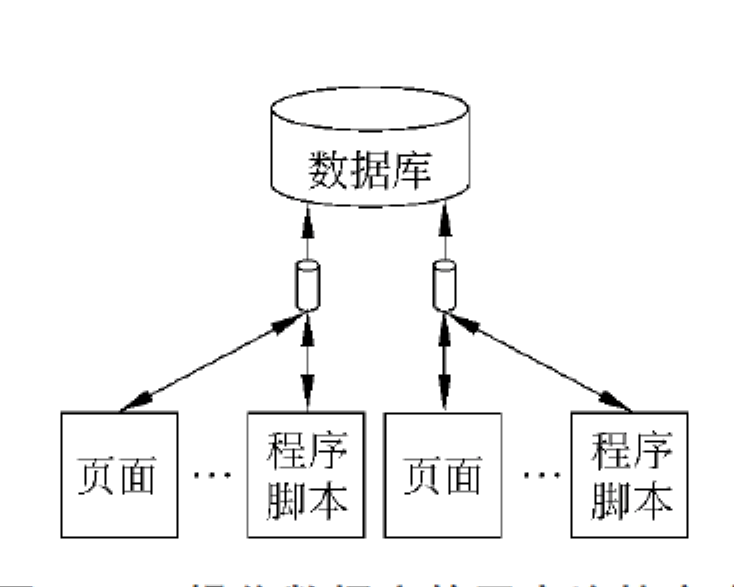


图 12-2 操作数据库的固定连接方式

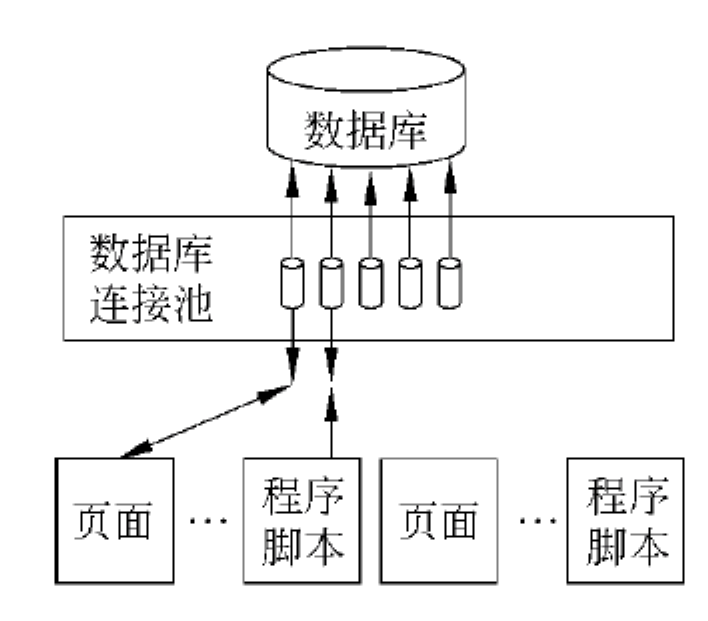


图 12-3 操作数据库的数据库连接池方式

SQL Server 数据提供程序自动为 ADO. NET 客户端应用程序提供连接池。对于 Web 应用程序，数据库连接池由 ASP. NET 自动管理，一般情况下不需要用户进行干预。ASP. NET 中管理连接池的是池进程。

每个数据库连接池中会保留一组活动的数据库连接。使用连接池，只要用户在连接对象上调用 Open 方法，池进程就会检查池中是否有可用的连接。如果某个池连接可用，

会将该连接返回给调用者，而不是打开新连接。应用程序在该连接上调用Close方法时，池进程会将连接返回到连接池的活动连接集中，而不是真正关闭连接。连接返回到池中之后，即可在下一个Open调用中重复使用。由此可见，使用数据库连接池，可以减少数据库连接打开和关闭的次数。

ADO.NET可同时保留多个连接池，每个连接池对应于一个数据库连接配置，数据库连接配置一般由连接字符串指定。

ASP.NET以如下的方式管理数据库连接池。

(1) 池的创建和分配

在应用程序初次打开一个数据库连接时，ASP.NET将根据完全匹配算法创建连接池，该算法将每个连接池与不同的连接字符串关联。打开新连接时，如果连接字符串并非与现有连接池完全匹配，则创建一个新连接池。

如果连接池的Min Pool Size属性在连接字符串中未指定或指定为零，池中的连接在一段时间不活动后将被关闭。但是，如果指定的Min Pool Size大于零，在应用程序结束之前，连接池不会被撤销(destroy)。非活动或空连接池的维护只需要极少的系统开销。

(2) 添加连接

当ASP.NET创建了一个连接池后，将创建多个连接对象并将其添加到该池中，以满足最小池的要求(参考Min Pool Size属性，默认值为0)。在应用程序的运行过程中，ASP.NET的池进程会根据需要向池中添加数据库连接，但是连接数不能超过指定的最大池(参考Max Pool Size属性，默认值为100)。

当应用程序请求SqlConnection对象时，如果连接池中存在可用的连接，将从池中获取该对象。连接在关闭或断开时释放回连接池中。

当应用程序请求连接时，如果池中连接数已达到最大池大小且不存在可用的连接，则该请求将会排队。然后，池进程尝试重新建立连接，直到达到超时时间(默认值为15秒)。如果池进程在连接超时之前无法满足连接请求，将引发异常。

(3) 移除连接

连接池进程定期扫描连接池，查找有没有应用程序通过调用连接对象的Close或Dispose方法关闭的未用连接，这些连接可供其他请求重新使用。因此，应用程序在使用完一个连接后显式调用Close和Dispose将其关闭是必要的，否则所占用的连接可能要到很长时间之后才能被重新使用。

如果连接长时间空闲或池进程尝试与服务器进行通信并检测到该连接与数据库服务器的连接已断开，连接池进程会将该连接从池中移除。

(4) 清除池

ADO.NET 2.0使用两种方法来清除连接池：ClearAllPools和ClearPool。ClearAllPools清除给定提供程序的所有连接池，ClearPool清除与特定连接字符串关联的连接池。

数据库连接池也不是毫无缺点，例如，连接池中可能存在着多个没有被使用的连接一直连接着数据库。默认情况下，ADO.NET中启用连接池，除非显式地禁用。根据需要，应用程序也可以不使用连接池。

通过对连接字符串中关键字的设置可控制连接池的使用。用于调整连接池行为的 ConnectionString 属性如表 12-1 所示。

表 12-1 可用于调整连接池行为的 ConnectionString 属性

属性名称	说　　明
Pooling	是否使用数据库连接池。当该属性值为 True 时，将从相应的连接池中取出连接。默认值为“True”
Connection Lifetime	当连接被释放回连接池后，创建时间将与当前时间进行比较，如果时间跨度(秒)超过 Connection Lifetime 指定的值，该连接将被清除。默认值为 0，将使池连接具有最大的超时期限
Enlist	默认值为“True”。当该属性值为 True 时，如果存在事务上下文，池进程将自动在当前事务上下文中登记连接
Max Pool Size	池中允许的最大连接数。默认为 100
Min Pool Size	池中维护的最小连接数。默认为 0

12.2 使用事务处理

事务是当前主流的数据库系统普遍采用的并发控制机制。

所谓事务，就是一个不可分割的数据库处理工作。事务中可能包含多个数据库操作(每个操作对应一条 insert、delete 或 update 语句)，这些数据库操作要么都发生，要么都不发生。图 12-4 以银行转账为例，说明事务处理的必要性。

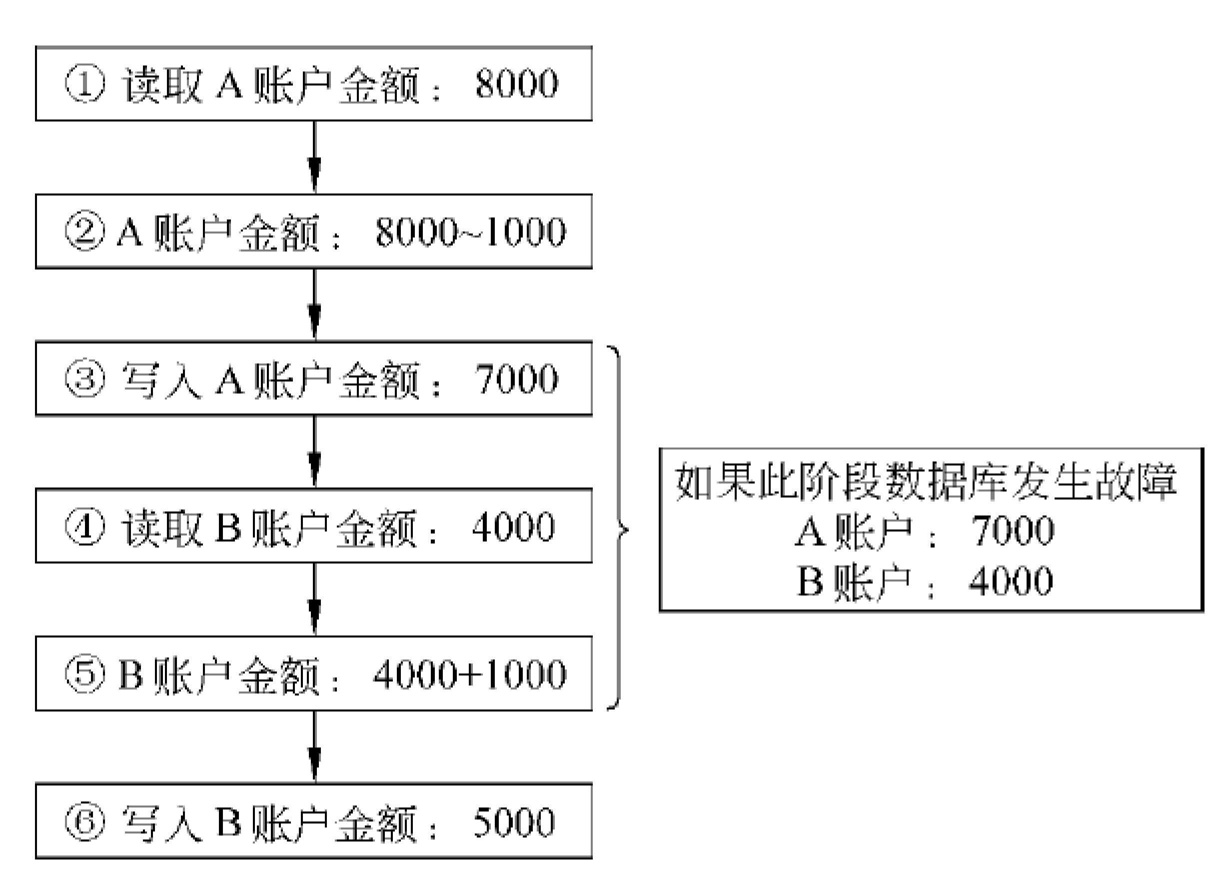


图 12-4 银行转账的处理流程

银行的账户数据都存储在数据库中。当需要从一个账户(A)向另一个账户(B)转账时，数据库操作总是要分为几个步骤来进行，如图 12-4 所示。如果没有事务机制，如果在这些步骤中间发生数据库故障(虽然概率很小，但不可避免)，转账则会出错。

好在当前主流的数据库管理系统都提供事务机制，能够保证多个数据库操作被作为

一个整体，或者全部执行，或者全部不执行。但是，数据库管理系统提供的只是事务控制机制，具体事务的控制（包括事务的开始、结束以及如何结束等），还是需要用户（或应用程序）自己来控制。当事务中所有的数据库操作完成时，只有当用户执行 Commit（提交）命令后，所有的操作才会被一次性地确认；如果在操作中出现数据库故障，或由用户执行 RollBack（回卷）命令，则前面已经执行的数据库操作都会自动恢复到事务开始时的状态。

事务控制往往被初级的编程者所忽视，其实它在程序开发中是很重要的。一旦需要，应用程序必须有效地控制事务。

在 ASP.NET 中，有两种方法实现事务处理。第一种方法是直接使用数据库本身的机制来管理事务。这种方法涉及较多的数据库相关知识，不同数据库的事务控制机制也有所不同，已经超出本书范围，不做详细介绍。第二种方法是使用 .NET 框架中的 SqlTransaction 类来定义事务，然后，就可以调用它的 Commit 或 Rollback 函数来控制事务了。这种方法虽然在后台仍然需要调用数据库本身的事务机制，但由于向用户屏蔽了事务处理的细节，使用起来更加方便；并且，使用这种方法还可以配合使用 .NET 框架提供的异常处理功能来获取系统异常。

本书主要讨论第二种方法，下面就以一个实例为基础加以说明。

首先，在本书的示例数据库 NetSchool 中创建一个账户表（Accounts），并插入几个账户记录（包括 li、gao 和 qi 等），相关 SQL 语句如下：

```
Use NetSchool
CREATE TABLE [Accounts] (
     [ID] [int] IDENTITY (1, 1) NOT NULL PRIMARY KEY ,
     [Name] [varchar] (50) NULL ,
     [Balance] [money] NULL
)
GO
Insert into [Accounts] ([Name],[Balance]) values ('li',10000)
Insert into [Accounts] ([Name],[Balance]) values ('gao',10000)
Insert into [Accounts] ([Name],[Balance]) values ('qi',10000)
GO
```

创建一个名为 TestTransaction 的网站。

在网站的默认主页上增加一个 GridView 控件。参阅 8.2.4 节的内容，为 GridView 控件增加一个数据源控件：取 Accounts 表的所有字段，按 ID 字段排序。

注意：数据源控件建立之后，系统还会为网站自动创建一个 Web 配置文件 Web.config。

再向主页增加一些其他操作控件，相关部分的代码如下：

```
<h2><span style="color: red">账户列表</span></h2>
<asp:GridView ID="GridView1" runat="server" AutoGenerateColumns="False" DataKeyNames="ID"
     DataSourceID="SqlDataSource1" Width="384px">
     <Columns>
          <asp:BoundField DataField="ID" HeaderText="ID" InsertVisible="False"
```

```
                ReadOnly="True"   SortExpression="ID"/>
            <asp:BoundField DataField="Name" HeaderText="姓名"
                SortExpression="Name"/>
            <asp:BoundField DataField="Balance" HeaderText="余额"
                SortExpression="Balance"/>
        </Columns>
</asp:GridView>
<asp:SqlDataSource ID="SqlDataSource1" runat="server" ConnectionString=
    "<%$ ConnectionStrings:SQLConnectionString %>"
    SelectCommand="SELECT * FROM [Accounts] ORDER BY [ID]">
</asp:SqlDataSource>
转账:从账户<asp:TextBox ID="Account_From" runat="server" Width="81px">
    </asp:TextBox>
转<asp:TextBox ID="Money" runat="server" Width="77px"></asp:TextBox>元
到账户<asp:TextBox ID="Account_To" runat="server" Width="74px">
    </asp:TextBox>。<br />
<asp:Button ID="Button1" runat="server" Text="执行" Width="78px"/><br />
<asp:Label ID="Label1" runat="server" Text=""></asp:Label>
```

当前页面可以执行,执行效果如图 12-5 所示。

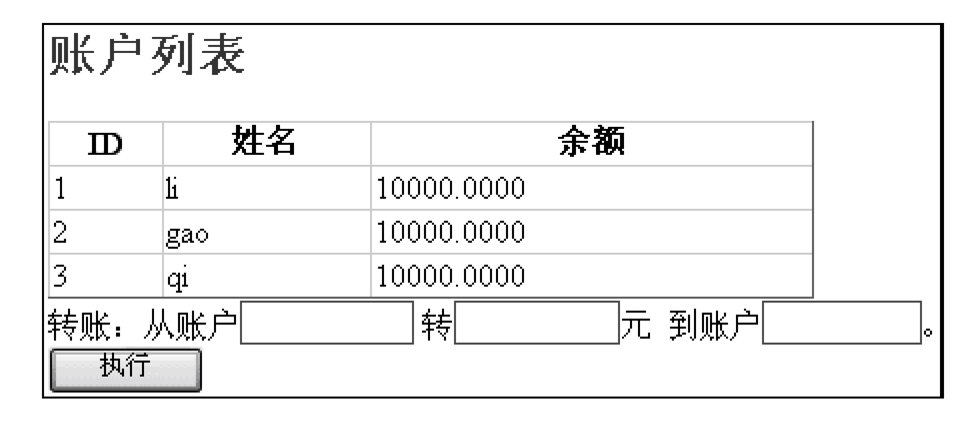


图 12-5 转账页面(一)

为"执行"按钮增加单击事件处理函数,代码如下:

```
protected void Button1_Click(object sender, EventArgs e)
{
    //创建连接对象并连接数据库
    string SQLConnectionString =
        ConfigurationManager.ConnectionStrings["SQLConnectionString"]
        .ConnectionString;
    SqlConnection myConnection = new SqlConnection(SQLConnectionString);
    myConnection.Open();

    //创建事务对象
    SqlTransaction myTrans = myConnection.BeginTransaction();
    //创建命令对象,并与事务对象关联
    SqlCommand myCommand = myConnection.CreateCommand();
    myCommand.CommandType = CommandType.Text;
    myCommand.Transaction = myTrans;
```

```
string str = "";
int i=0;

try
{
    //从源账户中减钱
    myCommand.CommandText = "Update Accounts set Balance
        =Balance-"+Money.Text+" where Name='"+Account_From.Text+"'";
    i = myCommand.ExecuteNonQuery();
    if (i == 1)
        str = str + "从账户" + Account_From.Text + "减钱成功。<br />";
    else
    {
        str = str + "从账户" + Account_From.Text + "减钱失败！<br />";
        myTrans.Rollback();
        str = str + "事务已回卷！<br />";
        return;
    }
    //向目标账户加钱
    myCommand.CommandText = "Update Accounts set Balance=Balance+" +
        Money.Text + " where Name='" + Account_To.Text + "'";
    i = myCommand.ExecuteNonQuery();
    if (i == 1)
        str = str + "向账户" + Account_To.Text + "加钱成功。<br />";
    else
    {
        str = str + "向账户" + Account_To.Text + "加钱失败！<br />";
        myTrans.Rollback();
        str = str + "事务已回卷！<br />";
        return;
    }
    //提交事务
    myTrans.Commit();
    GridView1.DataBind();
    str = str + "事务已提交！<br />";
    Console.WriteLine("Record is udated.");
}
catch (Exception ex)
{
    //如发生异常,事务回卷
    myTrans.Rollback();
    str = str + ex.Message;
    str = str + "数据库操作失败,事务已回卷！<br />";
}
```

```
        finally
        {
            //无论如何,关系数据库连接,并显示提示信息
            myConnection.Close();
            Label1.Text = str;
        }
    }
```

说明:在上述代码中,先分别创建事务对象和命令对象,事务对象与命令对象关联的方法是将事务对象指定为命令对象的 Transaction 属性值,如:

```
SqlTransaction myTrans = myConnection.BeginTransaction();
SqlCommand myCommand = myConnection.CreateCommand();
myCommand.Transaction = myTrans;
```

做了如此指定之后,经由该命令对象所执行的所有数据库操作命令,都可以进行事务控制处理。从上述代码中可以看到,当数据库操作发生可预见的错误时,如 myCommand.ExecuteNonQuery()的返回值不等于 1(表示无可操作数据或操作了多条数据),则将事务回卷;如果发生了不可预见的数据库操作错误,事务仍将回卷(在异常处理中);只有当所有操作都正确完成后,事务才会被提交。

重新执行页面。参照图 12-6,在文本框中输入相应的内容(从账户 qi 向账户 li 转 200 元),单击"执行"按钮,页面重新加载,可以看到两个相关账户中的余额已经改变,系统也给出了事务各步操作成功并最终提交的提示。

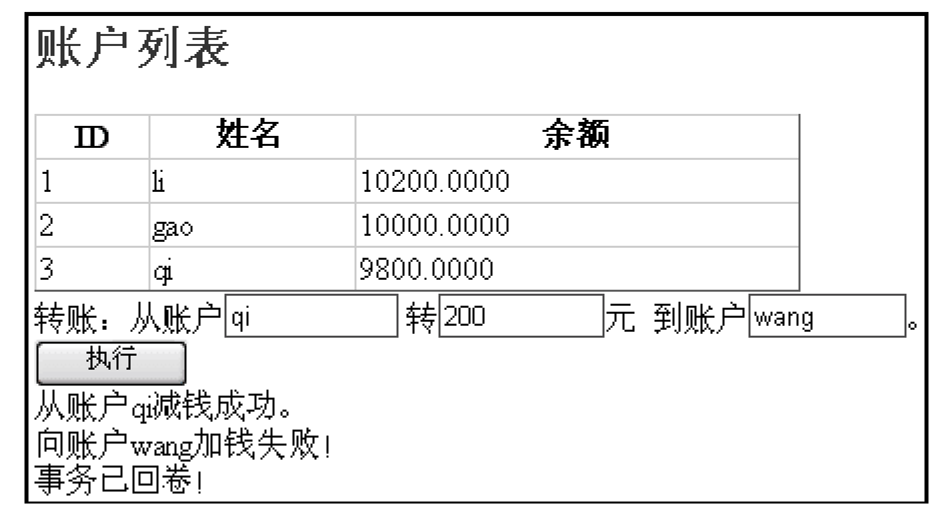
如果在执行期间数据库操作出错,或用户输入的账户名称有误,则数据库不会改动。在图 12-7 的输入中,目的账户名称有误(账户表中没有 wang 的信息),所以,虽然从源账户中减钱的操作已经成功,但事务回卷后,可以看到所有账户的余额都没有改变。

账户列表

ID	姓名	余额
1	li	10200.0000
2	gao	10000.0000
3	qi	9800.0000

转账:从账户 qi 转 200 元 到账户 li 。
执行
从账户qi减钱成功。
向账户li加钱成功。
事务已提交!

图 12-6 转账页面(二)



账户列表

ID	姓名	余额
1	li	10200.0000
2	gao	10000.0000
3	qi	9800.0000

转账:从账户 qi 转 200 元 到账户 wang 。
执行
从账户qi减钱成功。
向账户wang加钱失败!
事务已回卷!

图 12-7 转账页面(三)

12.3 高级 DataSet 技术

在本书前面的章节中已经介绍了 DataSet 对象和其内部的 DataTable 对象、DataRow 对象(详见 7.4 节),但介绍上述对象时都是从数据库中取数据。其实有时也许需要自行创建 DataSet 对象,并用程序向其中添加数据。

例如,在批量输入数据的过程中,如果所输入的数据存在着比较复杂的约束关系,一

个好的选择就是用程序创建 DataSet 对象，将数据输入到其内部的 DataTable 对象当中，从而利用 DataSet 本身所提供的约束机制来进行数据完整性的检查；当做好了各种检查及进一步的处理之后，再一次性地将这些数据导入到数据库当中。当然，这属于较高级的 DataSet 操纵技术，一旦熟悉了这项技术之后，在实现高级的数据应用时将会变得更加得心应手。

DataSet 是物理数据库（或物理数据库的一部分）在内存中的镜像，在物理数据库中能够实现的主要功能，在 DataSet 中基本都能实现。

学习过数据库相关知识的读者，相信对"部门-雇员表"的例子都不会陌生。这是一个经典的例子：一个单位可以有多个部门；每个部门可以有多个雇员；每个雇员只能属于一个部门；因此，在部门和雇员之间就形成了一个"一对多"的关系。在实现数据库时，需要分别建立两个表：部门表和雇员表。部门表和雇员表都有自己的主键，分别为部门编号和雇员编号；雇员表中也包含部门编号字段，说明该雇员属于哪个部门；在雇员表的部门编号字段上建立一个外键，完成参照完整性约束，使每个雇员必须属于一个已存在的部门。

在下面的例子中，将会直接在内存中创建一个 DataSet，在该 DataSet 中再创建两个 DataTable（而不是在数据库中创建表）。然后，分别在两个 DataTable 上建立数据列（字段）、约束、主键等，并输入示例数据。进一步在两个表之间建立一个外键和一个"关系"作为两个表之间的纽带。

创建一个名为 AdvancedDataSet 的网站。

在网站默认主页上增加两个 GridView 控件和两个 Button 控件，修改其属性，代码如下：

```
部门列表:<br />
<asp:GridView ID="Dept" runat="server">
</asp:GridView>
雇员列表:<br />
<asp:GridView ID="Emp" runat="server">
</asp:GridView>
<asp:Button ID="DS_Update" runat="server" Text="修改市场部编号"/>
<br />
<asp:Button ID="DS_Delete" runat="server" Text="删除研发部"/>
```

打开主页的隐藏代码页，为页类增加一个私有成员变量：

```
private static DataSet dataSet=null;
```

为页类增加一个成员函数，代码如下：

```
private void CreateDataSet()
{
    //创建一个 DataSet 实例
    dataSet = new DataSet();
```

```
////////////////////////////////////
//部门表
////////////////////////////////////
DataTable tblDept = new DataTable("Dept");

//创建部门编号字段
DataColumn newColumn;
newColumn = tblDept.Columns.Add("DeptNo", Type.GetType("System.Int32"));
newColumn.AllowDBNull = False;   //该字段不允许为空

//为部门编号创建唯一性约束
UniqueConstraint constraint =
   new UniqueConstraint("Unique_DeptNo", newColumn);
tblDept.Constraints.Add(constraint);

//为主键创建一个 columns 数组
DataColumn[] columnArray = new DataColumn[1];
//将部门编号字段加入到 columns 数组
columnArray[0] = newColumn;
//将数组增加为表的主键属性
tblDept.PrimaryKey = columnArray;

//为创建外键做准备
DataColumn deptNoColumn = newColumn;

//部门名称字段
newColumn = tblDept.Columns.Add("DName", Type.GetType("System.String"));
newColumn.AllowDBNull = False;
newColumn.MaxLength = 14;
newColumn.DefaultValue = "部门名称";

//部门所在地字段
newColumn = tblDept.Columns.Add("LOC", Type.GetType("System.String"));
newColumn.AllowDBNull = True;
newColumn.MaxLength = 13;

//向部门表插入数据
DataRow newRow;

newRow = tblDept.NewRow();
newRow["DeptNo"] = 10;
newRow["DName"] = "财务部";
newRow["LOC"] = "珠江路";
tblDept.Rows.Add(newRow);
```

```
newRow = tblDept.NewRow();
newRow["DeptNo"] = 20;
newRow["DName"] = "研发部";
newRow["LOC"] = "江宁";
tblDept.Rows.Add(newRow);

newRow = tblDept.NewRow();
newRow["DeptNo"] = 30;
newRow["DName"] = "市场部";
newRow["LOC"] = "珠江路";
tblDept.Rows.Add(newRow);

//将部门表加入到 DataSet 对象
dataSet.Tables.Add(tblDept);

////////////////////////////////////
//雇员表
////////////////////////////////////
DataTable tblEmp = new DataTable("Emp");

//创建字段
newColumn = tblEmp.Columns.Add("EmpNo", Type.GetType("System.Int32"));
newColumn.AutoIncrement = True;      //自动增长的字段
newColumn.AutoIncrementSeed = 1;
newColumn.AutoIncrementStep = 1;
newColumn.AllowDBNull = False;
newColumn.Unique = True;

newColumn = tblEmp.Columns.Add("EName", Type.GetType("System.String"));
newColumn.AllowDBNull = False;
newColumn.MaxLength = 10;

newColumn = tblEmp.Columns.Add("DeptNo", Type.GetType("System.Int32"));
newColumn.AllowDBNull = False;      // nulls not allowed

//为创建外键做准备
DataColumn empDeptNoColumn = newColumn;

newColumn = tblEmp.Columns.Add("Sal", Type.GetType("System.Int32"));
newColumn.AllowDBNull = False;
newColumn.DefaultValue = 0;

//输入数据
```

```
newRow = tblEmp.NewRow();
// EmpNo 字段由系统自动赋值(自动增长)
newRow["EName"] = "张三";
newRow["Sal"] = 800;
newRow["DeptNo"] = 10;
tblEmp.Rows.Add(newRow);

newRow = tblEmp.NewRow();
newRow["EName"] = "李四";
newRow["DeptNo"] = 20;
tblEmp.Rows.Add(newRow);

newRow = tblEmp.NewRow();
newRow["EName"] = "王五";
newRow["Sal"] = 1200;
newRow["DeptNo"] = 20;
tblEmp.Rows.Add(newRow);

newRow = tblEmp.NewRow();
newRow["EName"] = "赵六";
newRow["Sal"] = 1000;
newRow["DeptNo"] = 30;
tblEmp.Rows.Add(newRow);

// 将雇员表加入到 DataSet 对象
dataSet.Tables.Add(tblEmp);

// 创建外键约束
ForeignKeyConstraint fk = new ForeignKeyConstraint(
        "FK_EmpToDept", deptNoColumn, empDeptNoColumn);
fk.DeleteRule = Rule.Cascade;   // 级联删除
fk.UpdateRule = Rule.Cascade;   // 级联修改
tblEmp.Constraints.Add(fk);

// 声明 DataRelation 和 DataColumn 对象
System.Data.DataRelation dataRelation;
System.Data.DataColumn dataColumn1;
System.Data.DataColumn dataColumn2;

// 创建关系
dataColumn1 =
    dataSet.Tables["Dept"].Columns["DeptNo"];
dataColumn2 =
    dataSet.Tables["Emp"].Columns["DeptNo"];
```

```
        dataRelation =
            new System.Data.DataRelation(
            "R_EmpToDept",
            dataColumn1,
            dataColumn2);

        //将关系对象加入到 DataSet
        dataSet.Relations.Add(dataRelation);
}
```

该函数完成创建 DataSet 和 DataTable 的工作并生成数据。代码中有详细的注释，所以在此不再做进一步的说明，请读者仔细阅读上述代码。

再为页类增加一个成员函数，代码如下：

```
private void BindDataSet()
{
        //设置部门表数据源
        Dept.DataSource = dataSet.Tables["Dept"];
        Dept.DataBind();

        //设置雇员表数据源
        Emp.DataSource = dataSet.Tables["Emp"];
        Emp.DataBind();
}
```

该函数完成两个 GridView 控件的数据绑定。

修改 Page_Load()函数，代码如下：

```
protected void Page_Load(object sender, EventArgs e)
{
        if (!Page.IsPostBack)
        {
            CreateDataSet();
            BindDataSet();
        }
}
```

执行页面，效果如图 12-8 所示。

部门列表：

DeptNo	DName	LOC
10	财务部	珠江路
20	研发部	江宁
30	市场部	珠江路

雇员列表：

EmpNo	EName	DeptNo	Sal
1	张三	10	800
2	李四	20	0
3	王五	20	1200
4	赵六	30	1000

修改市场部编号

删除研发部

图 12-8　显示 DataSet 中的数据

从图 12-8 的显示中可以看出：

(1) 输入的数据都得以显示；

(2) 李四的 Sal 字段在输入数据时没有指定，但由于该字段指定了默认值 0，所以李四的 Sal 为 0；

(3) 所有雇员数据都没有指定 EmpNo 字段的值，但由于该字段被指定为自动增长的字段(种子为 1，增长步长为 1)，所以，各雇员的编号按插入顺序从 1 开始递增。

本示例已经演示了创建 DataSet 和 DataTable 并生成数据的方法。DataTable 一旦创建之后，就可以用操作数据库表类似的方法来操作它的数据，如查询、修改、删除等。

可以使用 DataTable 的 Rows 和 Columns 集合来访问 DataTable 中的内容，也可以根据包括搜索条件、排序顺序和行状态在内的特定条件，使用 Select 方法返回 DataTable 中数据的子集。此外，用主键值搜索特定行时，还可以使用 Find 方法。

DataTable 对象的 Select 方法返回一组与指定条件匹配的 DataRow 对象。Select 接受筛选表达式、排序表达式和 DataViewRowState 的可选参数。筛选表达式（例如 EName='李四'）根据 DataColumn 值标识要返回的行。排序表达式用于对结果集进行排序，可以是类似"EName ASC"的表达式。

为"修改市场部编号"按钮编写单击事件处理函数，代码如下：

```
protected void DS_Update_Click(object sender, EventArgs e)
{
    //取得 Dept 表
    DataTable dt = dataSet.Tables["Dept"];
    //查找符合条件的记录
    DataRow[] custRows = dt.Select("DeptNo = 30");
    //对符合条件记录进行处理
    for (int i = 0; i < custRows.Length; i++)
    {
        //修改部门编号
        custRows[0]["DeptNo"] = 50;
        custRows[0].AcceptChanges();
    }
    //重新绑定数据
    BindDataSet();
}
```

该函数先从 Dept 表中找到市场部所对应的记录("DeptNo=30")，将该记录 DeptNo 字段的值改为 50，然后重新为两个（注意，是两个）GridView 控件绑定数据。

重新执行页面，单击"修改市场部编号"按钮，效果如图 12-9 所示。



图 12-9　修改 DataSet 中的数据

可以看到，市场部的编号改成了 50，同时雇员赵六的 DeptNo 字段值也改为了 50。在程序中并没有显式地改变赵六的数据，但由于定义两表之间外键时指定了 UpdateRule 属性值为 Rule.Cascade(级联修改)，所以系统会自动将市场部所有雇员的部门编号进行相应的修改。

为"删除研发部"按钮编写单击事件处理函数，代码如下：

```
protected void DS_Delete_Click(object sender, EventArgs e)
{
```

```
        //取得 Dept 表
        DataTable dt = dataSet.Tables["Dept"];
        //查找符合条件的记录
        DataRow[] custRows = dt.Select("DeptNo = 20");
        //删除符合条件记录
        for (int i = custRows.Length - 1; i >= 0; i--)
            dt.Rows.Remove(custRows[i]);
        //重新绑定数据
        BindDataSet();
    }
```

该函数先从 Dept 表中找到研发部所对应的记录("DeptNo = 20")，然后执行 DataTable 的 Rows 属性的 Remove 方法删除该记录，最后再重新为两个 GridView 控件绑定数据。

部门列表:

DeptNo	DName	LOC
10	财务部	珠江路
30	市场部	珠江路

雇员列表:

EmpNo	EName	DeptNo	Sal
1	张三	10	800
4	赵六	30	1000

修改市场部编号

删除研发部

图 12-10　删除 DataSet 中的数据

重新执行页面，单击"删除研发部"按钮，效果如图 12-10 所示。

可以看到，在 Dept 表中研发部已被删除；同时，Emp 表中研发部的所有雇员同时也被删除。在程序中并没有显式地删除雇员数据，但由于在定义两表之间外键时指定了 DeleteRule 属性值为 Rule.Cascade(级联删除)，所以系统会自动将研发部的所有雇员删除。

习　　题

1. 为什么采用数据库连接池？
2. 数据库连接池在何时创建？
3. 试说明事务的概念。
4. 说明在 .NET 框架中进行事务控制的方法。
5. 当事务中所有操作都执行成功时，执行什么命令？当事务执行失败时，执行什么命令？它们各自的作用是什么？
6. 举例说明，在什么时候需要在程序中自行创建 DataSet 对象，并向其中添加数据？
7. 说明在 DataSet 对象中创建外键的方法。
8. 说明 DataTable 自动增长字段的特征及其定义方法。

第 13 章 chapter 13

畅想网络学院

畅想网络学院(在本章后面的内容中,可能简称“系统”或“本系统”)是一个网上教学与管理的平台。功能虽然不多,但却实现了以课程为核心的完整管理流程。学生可以下载所选课程的课件进行学习,并提出问题;教师可以上载课件、回答问题;管理人员可以对系统进行管理和配置。

畅想网络学院来源于一个实际的项目,但本身并不是一个实际的系统。为了适用于教学,畅想网络学院更注重把主要的控件都用上,主要的编程方法都用上,而并不一定是在最合适的场合选用了最合适的技术。即使是运用了的技术,为了教学的方便,和实用系统的实现方法也会有一定的差别。应用程序开发是一项创造性的劳动,从书上只能学到标准的实现方法。在实际系统的开发中,根据用户需要和编程习惯,会与书上的方法有很大的差别,需要加入编程人员许多创造性的劳动。

编程从来都不是学会的,而是“编”会的。从这一点来看,在经过了前面章节的学习之后,更不应该跳过本章,而是应该将本章的学习作为一个最佳的实践机会,也借此巩固前面章节的学习成果,达到融会贯通的效果。

13.1 系统总体设计

13.1.1 功能设计

“畅想网络学院”的主页界面如图 13-1 所示。

系统主页上提供个人日程表、新教室介绍、学院新闻、学校公告、友情链接、软件下载等一般性功能。

“畅想网络学院”的用户有三类,包括管理人员、教师和学生。三类人员采用统一的界面进行登录,登录后根据用户身份的不同提供不同的菜单功能。

1. “管理人员系统”的功能

- 教师管理:使用不同的技术,提供两种对教师信息进行管理的方法。
- 学生管理:使用不同的技术,提供三种对学生信息进行管理的方法。
- 课程管理:对学院所开课程进行管理。包括课程细节信息的管理,为课程选定任



图 13-1 “畅想网络学院”的主页

课教师，指定选修课程的学生等。

- 管理人员列表：使用不同的技术，提供三种对管理人员信息进行管理的方法。
- 修改密码：对本人的登录密码进行修改。

2. “教师系统”的功能

教师能够对所任教的课程进行管理。包括查看课程详细信息，为课程上载课件或删除已有课件，解答学生提出的问题等。

3. “学生系统”的功能

学生能够对所选课程进行学习。包括查看课程详细信息，下载、播放课件，提出问题等。

13.1.2 数据库设计

本系统使用 SQL Server 作为示例数据库，因此以下的内容是针对 SQL Server 数据库进行说明。

为完成上节所述功能，设计了如下的数据库表，以表格的形式加以介绍，如表 13-1～表 13-12 所示。

在表格的“类型”列中使用了一些类型名的缩写，如 V 代表 varchar 类型，D 代表 DateTime 类型，N 代表 decimal 类型。

在表格的“键”列中，“主”代表该字段为表的主键字段，“外”代表该字段为外键字段。外键字段没有说明是参照哪个表的哪个字段，但根据字段的含义应该不难看出。

在表格的“其他”列中，有★标记的表示该字段不能为空。

表 13-1　管理人员表（MANAGER）

字　段　名	中 文 说 明	类型	长度	键	其　　他
USERID	编号	V	20	主	★
USERNAME	姓名	V	20		★
PASSWORD	密码	V	50		
SEX	性别	V	4		
BIRTHDAY	出生日期	D			
DUTY	职务	V	20		
SPECIALTY	专业	V	50		
REMARK	备注	V	2000		

表 13-2　教师表（TEACHER）

字　段　名	中 文 说 明	类型	长度	键	其　　他
USERID	编号	V	20	主	★
USERNAME	姓名	V	20		★
PASSWORD	密码	V	50		
SEX	性别	V	4		
BIRTHDAY	出生日期	D			
RANK	职称	V	20		
SPECIALTY	专业	V	50		
REMARK	备注	V	2000		

表 13-3　学生表（STUDENT）

字　段　名	中 文 说 明	类型	长度	键	其　　他
USERID	编号	V	20	主	★
USERNAME	姓名	V	20		★
PASSWORD	密码	V	50		
SEX	性别	V	4		
BIRTHDAY	出生日期	D			
REGTIME	注册时间	D			
SPECIALTY	专业	V	50		
REMARK	备注	V	2000		

表 13-4　课程表(CLASS)

字　段　名	中文说明	类型	长度	键	其　　他
CLASSID	编号	V	20	主	★
CLASSNAME	课程名	V	50		★
REGTIME	开课时间	D			
SPECIALTY	专业	V	50		
REMARK	备注	V	2000		

表 13-5　教师-课程表(TEACHER_CLASS)

字　段　名	中文说明	类型	长度	键	其　　他
USERID	编号	V	20	主外	★
CLASSID	编号	V	20	主外	★
DUTY	责任	V	20		

表 13-6　学生-课程表(STUDENT_CLASS)

字　段　名	中文说明	类型	长度	键	其　　他
USERID	编号	V	20	主外	★
CLASSID	编号	V	20	主外	★
REGTIME	选课时间	D			
GRADE	成绩	N	5,2		

表 13-7　课件表(FILES)

字　段　名	中文说明	类型	长度	键	其　　他
CLASSID	编号	V	20	主外	★
FILENAME	文件名	V	200	主	★
FILETYPE	文件类型	V	200		
FILEURL	文件地址	V	200		
FILECONTENT	文件说明	V	2000		

表 13-8　提问与解答-主表(BBS)

字　段　名	中文说明	类型	长度	键	其　　他
BBSID	主题 ID	BIGINT		主	★IDENTITY (1,1)
CLASSID	课程号	V	20	外	★
TITLE	主题	V	100		★

续表

字 段 名	中 文 说 明	类型	长度	键	其 他
CONTENT	内容	V	1000		
USERTYPE	用户类型	V	1		T—教师 S—学生
USERID	编号	V	20		
USERNAME	姓名	V	20		
RESTYPE	用户类型(回复)	V	1		T—教师 S—学生
RESID	编号	V	20		
RESNAME	姓名	V	20		
BBSREAD	人气	INT			
BBSWRITE	回复数	INT			
BBSTIME	发表时间	D			
RESTIME	最新回复时间	D			

表 13-9 提问与解答-回复表(BBS_RESPONSE)

字 段 名	中 文 说 明	类型	长度	键	其 他
RESPONSEID	答复 ID	BIGINT		主	★IDENTITY (1,1)
BBSID	主题 ID	BIGINT		外	
RESTYPE	用户类型(回复)	V	1		T—教师 S—学生
RESID	编号	V	20		
RESNAME	姓名	V	20		
CONTENT	内容	V	1000		
RESTIME	回复时间	D			

表 13-10 新闻表(NEWS)

字 段 名	中 文 说 明	类型	长度	键	其 他
NEWSID	新闻 ID	INT	4	主	★IDENTITY (1,1)
TITLE	标题	V	200		
BODY	内容	TEXT			
NEWSDATE	日期	D			

表 13-11 字段名表(SYS_FIELD)

字 段 名	中 文 说 明	类型	长度	键	其 他
TABLE_NAME	表名	V	30	主	★
COLUMN_NAME	列名	V	30	主	★

续表

字 段 名	中文说明	类型	长度	键	其 他
COLUMN_ID	列号	INT	4		
DATA_TYPE	数据类型	V	1		
DATA_LENGTH	列宽	INT	4		
COLUMN_KIND	列说明	V	60		
DICK	数据字典	V	2		
DISP	是否显示	V	1		

表 13-12 菜单表(MENU)

字 段 名	中文说明	类型	长度	键	其 他
USERTYPE	用户类型	CHAR	1	主	★
TREEID	功能 ID	INT	4	主	★
TITLE	菜单标题	V	50		
DESN	描述	V	200		
PARENTID	父功能 ID	INT	4		
URL	功能 URL	V	200		
TARGET	目标	V	50		

本节只是对数据库设计做一个简单说明，读者也可以按下节的方法建立示例数据库后，到数据库管理工具中查看实际的数据库设计。

13.1.3 示例数据库的建立

示例数据库中包含了畅想网络学院运行所必需的数据，因此，在运行畅想网络学院之前必须正确建立示例数据库。

建立示例数据库有两种方法，第一种方法是按照 13.1.2 小节所介绍的有关数据库设计的内容手工建立。在 SQL Server 2000 中创建一个名为 NetSchool 的数据库；按照 13.1.2 小节所述内容，逐个建立数据库表，并输入示例数据。

第二种方法是附加数据库。将 NetSchool_Data.MDF 和 NetSchool_Log.LDF 两个文件复制到硬盘适当的目录下面，然后在 SQL Server 2000 的企业管理器中附加数据库。方法为：在企业管理器中选择相应的数据库服务器，在“数据库”标签上单击鼠标右键，按如图 13-2 所示选择“附件数据库”。“附加数据库”对话框如图 13-3 所示。

选择要附加数据库的文件，指定附加后的数据库名和数据库所有者，单击“确定”按钮，完成附加数据库的操作。

畅想网络学院使用数据库用户 sa 来访问示例数据库，因此，无论使用哪种方法建立示例数据库，都还需要在畅想网络学院网站建立后(建立网站的内容见 13.1.2 小节)对

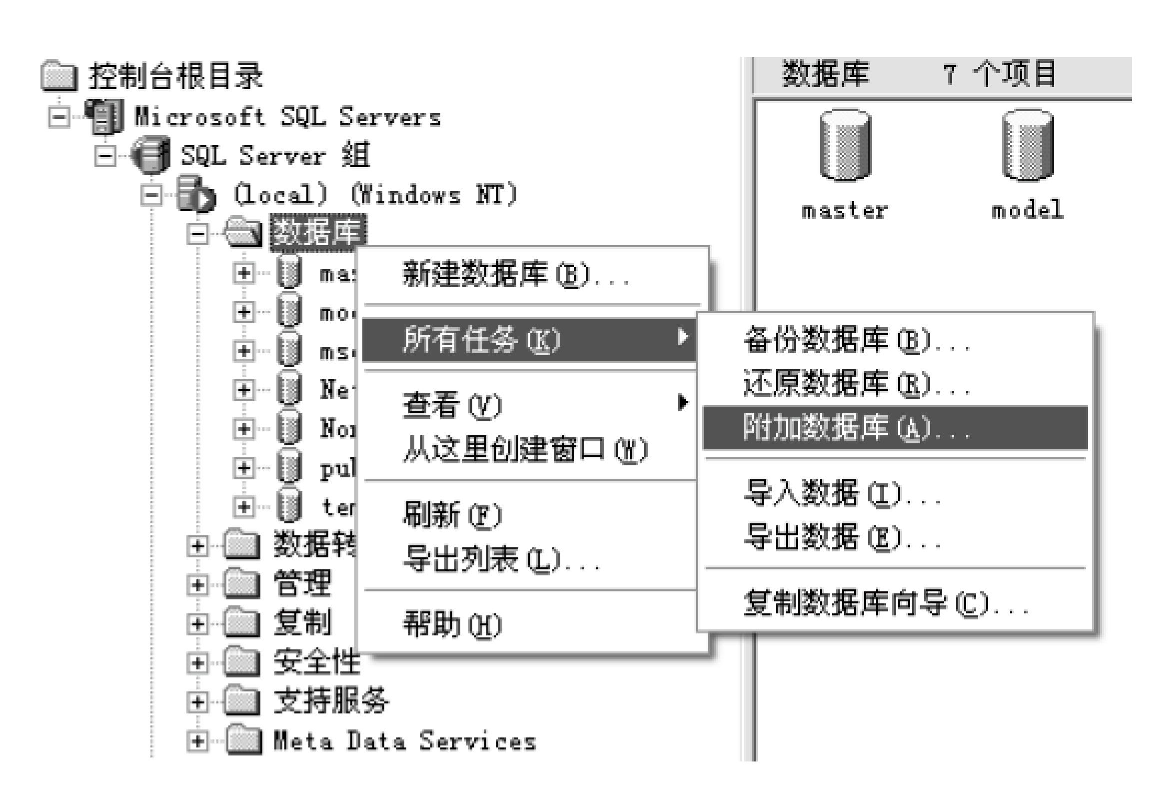


图 13-2 附加数据库

附加数据库 — (local)
要附加数据库的 MDF 文件：(M)
F:\NetworkAppCode\DB\NetSchool_Data.MDF
验证(V)

原文件名	当前文件位置
NetSchool_Data.MDF	F:\NetworkAppCode\DB\NetSchool_Data.MDF
NetSchool_Log.LDF	F:\NetworkAppCode\DB\NetSchool_Log.LDF

附加为(A)：NetSchool
指定数据库所有者：(S) sa
确定 取消 帮助

图 13-3 “附加数据库”对话框

其 Web 配置文件(Web.config)进行修改。

可以用任何文本编辑器打开 Web.config 文件，打开数据库连接串 SQLConnectionString 的配置部分，在连接串中配置正确的服务器名(data Source 属性)，设置正确的 sa 用户口令(pwd 属性)。

13.1.4 网站项目的创建

为畅想网络学院创建一个称为 NetSchool 的网站。

为网站创建一个 Web 配置文件 Web.config。在 Web.config 中配置一个称为 SQLConnectionString 的连接字符串，其值可能为：

```
data Source=(local);database=NetSchool;user id=sa;pwd=abc
```

在系统实际运行时，用数据库服务器的机器名代替(local)，用 sa 用户的实际口令代替 abc。

为网站创建一个全局应用程序类 Global.asax，将其中的全局事件处理函数 Application_Start 改为如下代码：

```
void Application_Start(object sender, EventArgs e)
```

```
{
    //在应用程序启动时运行的代码
    Application["ApplicationName"] = "畅想网络学院";
    Application["PageSize"] = "12";
}
```

其中：应用程序变量 ApplicationName 存储系统名称，系统中所有需要显示系统名称的地方都从该变量中取值。因此，如果需要更换系统名称，只需要修改此处应用程序变量的赋值即可。

应用程序变量 PageSize 存储列表显示数据时每页的默认行数。系统中需要列表显示数据时，会动态地从该变量中取值，并设置每页的行数。

在实际的应用系统中，这一类的配置项一般是存储在数据库中的，并为系统管理员提供配置工具对其进行配置管理。本系统为简单起见，在此对其直接赋值。

网站的目录设计如下。

(1) 将与主页有关的一些公共页面放在网站根目录下。

(2) 创建一个 ASP.NET 保留文件夹 App_Code，用于存放系统的各公用类。

(3) 创建 common 文件夹，用于存放系统的所有用户控件和层叠样式表。

(4) 创建 images 文件夹，用于存放系统使用的所有图片文件。

(5) 创建专用的 js 文件夹，用于存放系统使用的 JavaScript 自定义控件。

(6) 创建 Manage 文件夹，用于存放管理人员系统的各功能页面。

(7) 创建 Teacher 文件夹，用于存放教师系统的各功能页面，其中有些页面与学生系统共用。

(8) 创建 Student 文件夹，用于存放学生系统的各功能页面(当前为空，还没有专为学生开发的功能)。

(9) 创建 Uploads 文件夹，用于存放为各课程上载的课件文件。开始时该文件夹为空。

13.2 系统体系结构的设计与实现

畅想网络学院系统采用 .NET Web 应用程序设计常用的四层结构，即数据库层、数据访问层、业务逻辑层和表示层，如图 13-4 所示。

数据库层是整个系统的基础，它保存并维护系统的所有数据。其设计在 13.1 节已经介绍，这里不再重复。

数据访问层封装访问数据库的各种通用操作，如连接数据库、数据的读/写操作、断开数据库连接等。该层由 Database.cs 文件中的 Database 类实现。

业务逻辑层调用数据访问层的功能，为上层页面提供数据服务。它的作用是对上层屏蔽数据库操作的细节，使上层只关心数据之间的逻辑关系，从而简化数据访问的接口。该层包括 BBS、Files、Menu、News、QrySet、Student、User 等一些数据库逻辑操作类。

表示层实现系统的具体功能，包括各功能页面(.aspx)、用户控件(.ascx)、层叠样式

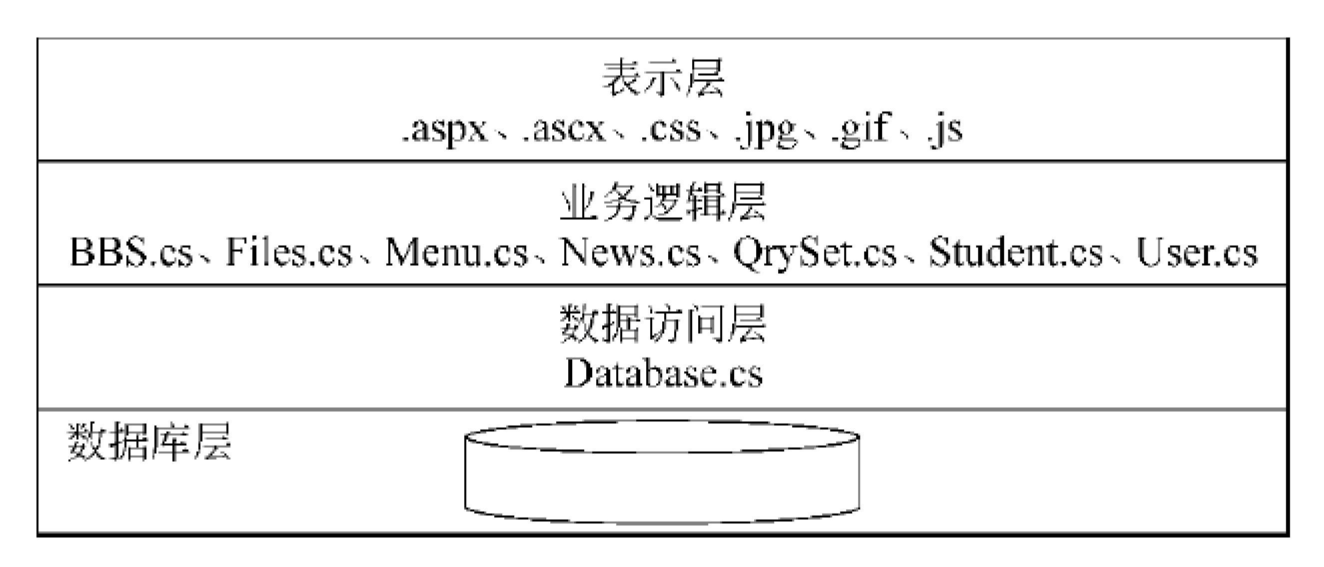


图 13-4 畅想网络学院系统的体系结构

表(.css)、图片(.jpg、.gif)和 JavaScript 控件(.js)等。

13.2.1 数据访问层的实现

将所有与数据库访问有关的操作集成在 Database.cs 中，方法如下：

在保留文件夹 App_Code 中创建一个类 Database.cs。为该类说明一个私有的 SqlConnection 对象 conn。

```
//数据库连接
private SqlConnection conn;
```

所有对数据库的访问都通过 conn 来进行。但由于 conn 是一个私有成员，其他的类不能直接使用它，只有通过 Database 类的成员函数才能使用它，这就保证了对数据库的所有(通过数据源控件的除外)底层操作都由 Database 类来提供。

在 Database 类中实现了以下成员函数。

(1) 打开数据库连接。

功能：如果 conn 为空，则创建；如果尚未连接到数据库，则连接；如果已经连接，则不做任何操作。

```
public void Open()
{
    if (conn == null)
    {
        conn = new
SqlConnection(ConfigurationManager.ConnectionStrings
["SQLConnectionString"]
.ConnectionString);
    }
    if (conn.State == ConnectionState.Closed)
    {
        try
        {
            //打开数据库连接
            conn.Open();
```

```
            }
            catch (Exception ex)
            {
                //抛出异常
                throw new Exception(ex.Message, ex);
            }
            finally
            {
                //关闭已经打开的数据库连接
            }
        }
    }
```

（2）关闭数据库连接。

功能：如果 conn 不为空，且状态为“打开”，则将其关闭。

```
public void Close()
{
    //判断连接是否已经创建
    if (conn != null)
    {
        //判断连接的状态是否打开
        if (conn.State == ConnectionState.Open)
        {
            conn.Close();
        }
    }
}
```

（3）释放资源。

功能：释放 conn 所占用的资源。

```
public void Dispose()
{
    //确认连接是否已经关闭
    if (conn != null)
    {
        conn.Dispose();
        conn = null;
    }
}
```

（4）运行 SELECT 语句，将查询结果以 SqlDataReader 对象返回。

```
/// <param name= "sqlText"> SQL语句< /param>
```

```
/// <returns> SqlDataReader 对象</returns>
public SqlDataReader RunSQLtoDataReader(string sqlText)
{
    //获得并打开数据库连接
    Open();

    //创建 Command
    SqlCommand command = new SqlCommand(sqlText, conn);

    ///定义 DataReader
    SqlDataReader dr = null;
    try
    {
        //读取数据
        dr = command.ExecuteReader(CommandBehavior.CloseConnection);
    }
    catch (SqlException ex)
    {
        //抛出异常
        throw new Exception(ex.Message, ex);
    }
    ///返回 DataReader
    return dr;
}
```

(5) 运行 SELECT 语句，将查询结果以 DataTable 对象返回。

```
/// <param name="sqlText"> SQL 语句</param>
/// <returns> DataTable 对象</returns>
public DataTable RunSQLtoDataTable(string sqlText)
{
    //获得并打开数据库连接
    Open();

    SqlDataAdapter adapter = new SqlDataAdapter();
    adapter.SelectCommand = new SqlCommand(sqlText, conn);

    DataSet dataSet = new DataSet();
    adapter.Fill(dataSet, "NetSchool");
    DataTable dataTable = dataSet.Tables["NetSchool"];

    Close();
    return dataTable;
```

```
}
```

(6) 执行数据库修改命令。

功能：执行包括 update、insert 和 delete 在内的，除 select 之外的数据管理命令。

```
/// <param name="cmdText">命令字符串</param>
public void RunCmd(string cmdText)
{
    try
    {
        Open();

        System.Data.SqlClient.SqlCommand command =
          new System.Data.SqlClient.SqlCommand();
        command.Connection = conn;
        command.CommandText = cmdText;
        command.ExecuteNonQuery();
    }
    finally
    {
        Close();
    }
}
```

13.2.2 业务逻辑层的实现

业务逻辑层由 BBS、Files、Menu、News、QrySet、Student、User 等业务逻辑类组成。每个业务逻辑类对一类业务数据进行操作，它们都调用 Database 类所提供的功能。

本节对上述各类不做一一介绍，仅以 User 类为例加以说明。User 类的源代码如下：

```
using System;
using System.Data;
using System.Configuration;
using System.Web;
using System.Web.Security;
using System.Web.UI;
using System.Web.UI.WebControls;
using System.Web.UI.WebControls.WebParts;
using System.Web.UI.HtmlControls;
using System.Data.SqlClient;

/// <summary>
/// 为用户登录功能实现的数据库操作类
/// </summary>
public class User
```

```
{

    public User()
    {
        ///
    }

    /// <summary>
    /// 加密函数
    /// </summary>
    private string Encrypt(string password)
    {
        //示例程序,不加密,直接返回原字符串
        return password;
    }

    /// <summary>
    /// 用户登录验证
    /// </summary>
    /// <param name="sUserType">用户类型</param>
    /// <param name="sUserId">用户号</param>
    /// <param name="sPassword">口令</param>
    /// <returns>根据登录信息取得的结果集</returns>
    public SqlDataReader GetUserLoginBySQL(string sUserType, string sUserId, string sPassword)
    {
        //定义 SQL 语句
        string cmdText = "select *  from ";
        switch (sUserType)
        {
            case "教师":
                cmdText = cmdText + "teacher";
                break;
            case "学生":
                cmdText = cmdText + "student";
                break;
            case "管理人员":
                cmdText = cmdText + "manager";
                break;
            default:
                return null;
        }
        cmdText = cmdText + " WHERE UserId = '" + sUserId + "'"
            + " AND Password = '" + Encrypt(sPassword) + "'";
```

```
            //取数据
            Database db = new Database();
            SqlDataReader dr = db.RunSQLtoDataReader(cmdText);

            //返回 DataReader
            return dr;
        }

        /// <summary>
        /// 修改用户口令
        /// </summary>
        /// <param name="sUserType">用户类型</param>
        /// <param name="sUserId">用户号</param>
        /// <param name="sPassword">新用户密码</param>
        public void ChangeUserPwd(string sUserType, string sUserId, string sPassword)
        {
            //定义 SQL语句
            string cmdText = "update ";
            switch (sUserType)
            {
                case "教师":
                    cmdText = cmdText + "teacher";
                    break;
                case "学生":
                    cmdText = cmdText + "student";
                    break;
                case "管理人员":
                    cmdText = cmdText + "manager";
                    break;
                default:
                    return;
            }
            cmdText = cmdText + " set PASSWORD= '" + Encrypt(sPassword) + "'WHERE UserId = '" + sUserId + "
'";

            //修改数据库
            Database db = new Database();
            db.RunCmd(cmdText);
        }
    }
```

说明：在实际的应用中，即使是在数据库中，也不应该保存用户口令的明文，需要对其进行加密。User类中的Encrypt()函数完成此功能，但由于本系统仅仅是示例程序，为了降低复杂性，没有真正地加密，而是直接返回了原字符串(口令的明文)。在实际应

用中，读者可以根据需要自行设计实现自己的加密算法。

GetUserLoginBySQL()函数在系统登录和修改用户口令时被调用，根据用户类型及输入的用户名和密码，到数据库中查询用户信息。将查询所得的结果集(可能为空)返回给调用者，供判断登录信息是否合法。

ChangeUserPwd()函数在修改用户口令时被调用。当判断完用户原口令的正确性后，执行此函数将用户的口令改为新口令。

13.2.3 表示层的实现

表示层实现系统的具体功能，包括各功能页面(.aspx)、用户控件(.ascx)、层叠样式表(.css)、图片(.jpg、.gif)、JavaScript控件(.js)等。

1. 默认主页的功能

默认情况下，使用ASP.NET生成的网站，其默认主页为Default.aspx。

本系统的默认主页没有显示内容，但在它的隐藏代码中包含分支逻辑：如果用户未登录，则将链接重定向到rightframe.aspx，该页面是系统的实际主页，包含登录界面；如果用户已经登录，则重定向到mainframe.aspx，该页面包含登录后的系统菜单和系统主页。

Default.aspx.cs中的Page_Load()函数代码如下：

```
protected void Page_Load(object sender, EventArgs e)
{
    //检查Cookies对象
    //略：有关Cookie的操作见13.5节

    //检查会话对象
    if (Session["UserID"] != null)
    {
        //用户已登录，加载带菜单的页面
        Response.Redirect("mainframe.aspx");
    }
    else
    {
        //用户未登录，显示网站主页面
        Response.Redirect("rightframe.aspx");
    }
}
```

2. 右侧框架——真正的系统主页

rightframe.aspx为系统的实际主页。它使用HTML框架元素，将页面分为上下两个frame，如图13-5所示，上面的frame显示系统标题等信息，框架名为title；下面的

frame 在初始时用于显示系统主页各要素；系统运行中被用做系统主工作区，框架名为 mainwork。

title： 显示 title. aspx。
mainwork： 初始时显示 mainpage. axpx。

图 13-5 系统主页结构

rightframe. aspx 的关键部分代码如下：

```
<head runat="server">
    <title><%=Application["ApplicationName"]%></title>
</head>
<frameset rows="70, * " frameborder="NO" border="0" framespacing="0">
  <frame name="title"  src="title.aspx"scrolling="NO" noresize >
  <frame name="mainwork"  src="mainpage.aspx" scrolling=auto noresize>
</frameset>
```

title. aspx 的实现方法在 4. 5. 2 小节中已经介绍，这里不再赘述。下面介绍 mainpage. aspx 的实现方法。

3. 主页面

mainpage. aspx 用于显示主页的各功能要素，如图 13-1 的下半部分所示。可以看出各功能要素在布局上分为左、中、右三个部分，左侧为个人日程表、用户登录(登录后为“我的书桌”)和新教室介绍等要素；中部为系统介绍；右侧为学院新闻、学院公告、友情链接、软件下载等要素。页面的下部为系统统一的 copyright 信息。

用一个包含 1 行 3 列的 table 来实现布局，各要素用用户控件来实现。部分用户控件在 10.3 节已经介绍，这里不再赘述。

mainpage. aspx 的代码如下：

```
<%@  Page Language="C#" AutoEventWireup="True" CodeFile="mainpage.aspx.cs" Inherits="
mainpage"%>

<%@ Register Src="common\introduction.ascx" TagName="introduction"
TagPrefix="uc1"%>
<%@ Register Src="common\newclassroom.ascx" TagName="newclassroom"
TagPrefix="uc2"%>
<%@ Register Src="common\OfficeCalendar.ascx" TagName="OfficeCalendar"
```

```
TagPrefix="uc3"%>
<%@ Register Src="common\login.ascx" TagName="login" TagPrefix="uc4"%>
<%@ Register Src="common\ActiveOp.ascx" TagName="ActiveOp" TagPrefix="uc5"%>
<%@ Register Src="common\NewsUC.ascx" TagName="NewsUC" TagPrefix="uc6"%>
<%@ Register Src="common\copyright.ascx" TagName="copyright" TagPrefix="uc8"%>

<html xmlns="http://www.w3.org/1999/xhtml" >
<head runat="server">
    <title></title>
    <link href="common\StyleSheet.css" rel="stylesheet" type="text/css"/>
</head>
<body bottommargin=0 topmargin=2 leftmargin=0 rightmargin=0 style="overflow-x:hidden;">
    <form id="form1" runat="server" target="mainwork">
    <div>
        <table width="100% " border="0" class="all" align=left>
            <tr>
                <td width="214" class="right" valign=top
                        background="images/39.jpg">
                    <uc3:OfficeCalendar ID="OfficeCalendar1" runat="server"/>
                    <br />
                    <uc4:login ID="Login1" runat="server"/>
                    <br />
                    <uc2:newclassroom ID="newclassroom" runat="server"/>
                </td>
                <td class="right" valign=top>
                    <uc1:introduction ID="introduction" runat="server"/>
                    <uc8:copyright ID="copyright" runat="server"/>
                </td>
                <td width="214" valign=top background="images/39.jpg">
                    <uc6:NewsUC id="MyNewsUC" runat="server"></uc6:NewsUC>
                    <br/>
                    <uc5:ActiveOp ID="ActiveOp" runat="server"/>
                </td>
            </tr>
        </table>
    </div>
    </form>
</body>
</html>
```

页面上半部分为对用户控件的集中声明，下半部分为对各用户控件的引用。由此可以看出：像 mainpage.aspx 这样的复杂页面，在使用了用户控件之后，代码变得非常清晰。

4. 主框架页面

当系统登录后,系统主页面为 mainframe. aspx。该页面是采用一些高级页面技术实现的主框架,由三个 div 组成,如图 13-6 所示。

(1) div1(图中灰色阴影部分)

其 id 为 menu_div,包含一个称为 menu_iframe 的 iframe 对象,在其中调用 menu. aspx 页面显示菜单。该 div 平时是不显示的,被激活时才显示。onmouseout = "switchSysBar()"表示当鼠标移出该 div 时,执行函数功能,菜单被重新隐藏。

(2) div2(图中网格部分)

其 id 为 menu_bar,包含一个 table,onmouseover="switchSysBar( )"表示当鼠标进入该 div 区域时,执行函数功能,激活菜单 div(div1)。

(3) div3(图中右侧空白部分)

其 id 为 rightframe_div,包含一个称为 rightframe 的 iframe 框架对象,该框架中显示 rightframe. aspx 页面。

rightframe. aspx 页面初始时用于显示系统主页,其下部用做主工作区。

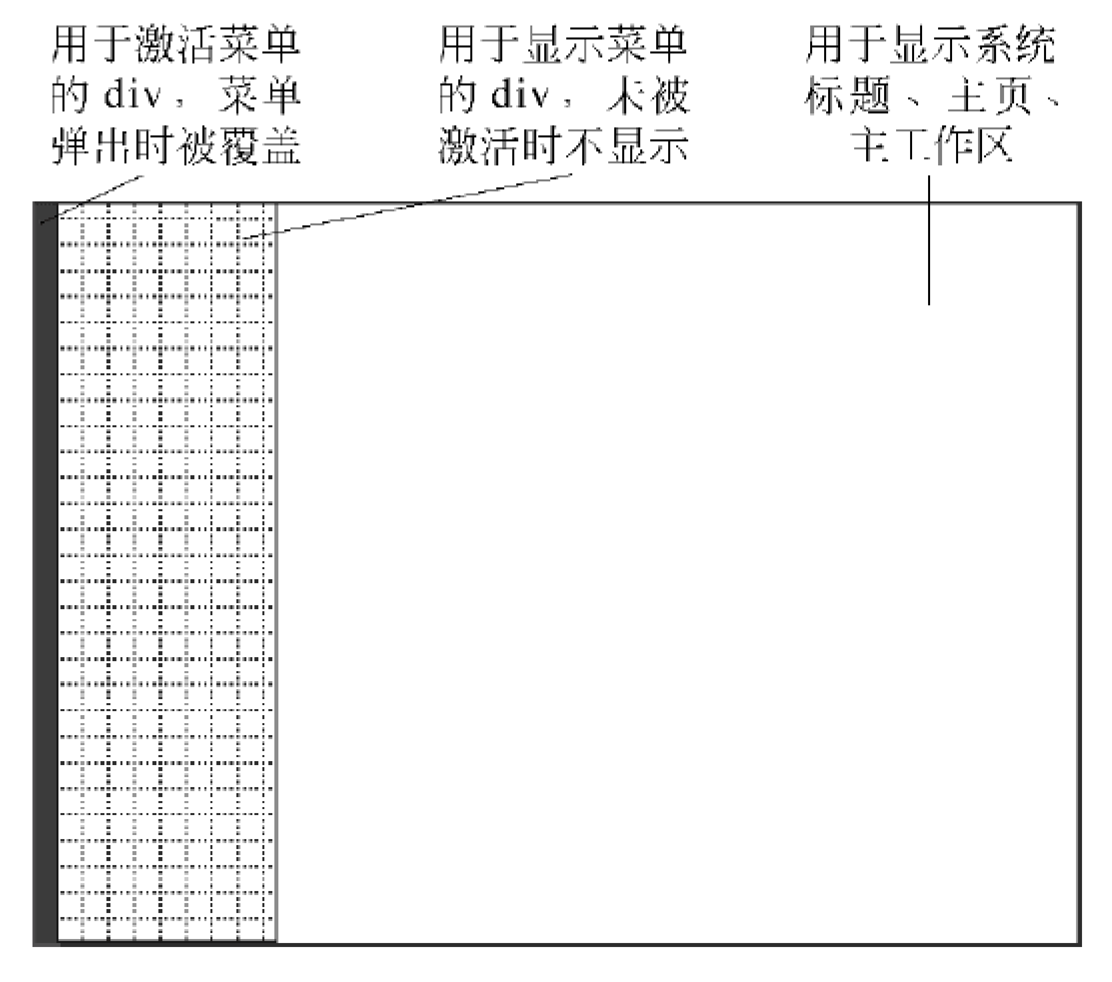


图 13-6　系统主框架

为了便于理解,将主框架页面的代码进行了简化,去掉了一些不重要的属性设置,主要代码如下:

```
<script language="JavaScript" type="text/JavaScript">
    //设置窗口的位置和大小
    window.moveTo(0,0);
    window.resizeTo(screen.availWidth,screen.availHeight);
</script>

<html>
<head id="Head1" runat="server">
```

```
    <title><%=Application["ApplicationName"]%></title>
<script language="JavaScript" type="text/JavaScript">
function switchSysBar()      //激活或隐藏菜单
{
    if(menu_div.style.display=='none')
    {
        menu_div.style.display='block';
        menu_bar.style.display='none';
    }else
    {
        menu_div.style.display='none';
        menu_bar.style.display='block';
    }
}
</script>
</head>
<body>
    <!--div1:用于显示菜单内容-->
    <div id="menu_div" style="position:absolute; width:180px; z-index:3;
            display:none" onmouseout="switchSysBar()">
        <iframe name="menu_iframe" src="menu.aspx">
        </iframe>
    </div>

    <!--div2:用于激活菜单-->
    <div id="menu_bar" style="position:absolute; width:2%; display:block"
            onmouseover="switchSysBar()">
        <table>
            <tr>
                <td>
                    ◀
                </td>
            </tr>
        </table>
    </div>

    <!--div3:包括系统主工作区 -->
    <div id="rightframe_div" style="position:absolute; width:98%; height:100%;
            z-index:1; left: 2%; top: 0px; display:block">
        <iframe marginwidth="0" marginheight="0" name="rightframe"
            scrolling="auto" src="rightframe.aspx"
            style="HEIGHT:100%;WIDTH:100%;">
        </iframe>
    </div>
```

```
</body>
</html>
```

说明：因为是系统主界面，我们希望其内容能完整展示，所以页面的第一部分代码是设置窗口的位置和大小。

switchSysBar()函数在移入 div2 和移出 div1 时被调用，其功能是先判断菜单 div 是否已经显示，再根据判断结果对 div1 进行隐藏和显示。

5. 尚未完成页面

由于本系统只是一个用于教学的示例系统，其中有些功能还未完成或没必要完成，有些超链接指向的页面还没有实现，所以需要一个尚未完成页面来作为这些超链接的指向，并可简单地显示一些提示信息，本系统由 undone.aspx 来完成此功能。

该页面的调用形式为“undone.aspx? mess=提示信息”。

整个页面仅包含一个 Label 控件和一句说明文字：

```
<asp:Label ID="Message" runat="server" Font-Names="华文新魏"
    Font-Size="36pt" ForeColor="Blue"></asp:Label><br />
抱歉，此功能尚未完成。
```

在页面的 Page_Load 事件中，从 Request 对象中取得参数，在 Label 上显示。

```
protected void Page_Load(object sender, EventArgs e)
{
    if (!Page.IsPostBack)
    {   ///显示信息
        Message.Text = Request.Params["mess"].ToString();
    }
}
```

在下面几节中继续介绍表示层各具体功能的实现，其中 13.3 节～13.6 节介绍一些通用模块，13.7 节～13.10 节介绍一些具体功能。

13.3 系统登录

与大多数应用系统一样，当用户想要进入本系统时，首先需要在系统的登录界面中输入用户名、密码和验证码，以验证用户身份。

应用系统永远也不能要求用户不犯错误，一个实用的系统应该具有验证用户输入正确与否的功能，以避免用户的误操作给用户和系统造成严重损害。

以往完成这些工作需要大量的自定义代码，使用 ASP.NET 提供的验证控件可以大大地简化这一过程。关于 ASP.NET 验证控件的详细介绍请参阅 5.6 节内容。

本系统在用户登录时使用了 RequiredFieldValidator 控件，以保证用户登录信息的输入不为空。在“修改口令”功能中使用了 CompareValidator 控件，以保证两次输入的新

密码一致,详见 13.6 节。

本系统以用户控件的形式实现系统登录界面。

Login.ascx 是一个比较完善的用户控件。页面上不但包含 TextBox、Button 等一般控件,还包含 RequiredFieldValidator 这样的验证控件。该用户控件中还包含完整的验证码图片的生成及验证过程。

1. 根据系统是否登录,显示不同的界面

Login.ascx 主要包含两部分界面,分别放置在两个 Panel 控件之上,一个显示登录界面,一个显示登录后的个人相关信息("我的书桌")。

```
<asp:Panel ID="Panel1" runat="server">
    登录界面(登录之前)
</asp:Panel>
<asp:Panel ID="Panel2" runat="server">
    我的书桌(登录之后)
</asp:Panel>
```

在页面的 Page_Load 函数中对系统是否已经登录进行判断,从而确定显示两个 Panel 控件中的哪一个。

```
protected void Page_Load(object sender, EventArgs e)
{
    //检查系统是否已经登录,确定显示哪个 Panel
    if (Session["UserID"] !=null)
    {
        //已经登录,显示"我的书桌"
        this.Panel1.Visible = False;
        this.Panel2.Visible = True;
    }
    else
    {
        //未登录
        this.Panel2.Visible = False;
        this.Panel1.Visible = True;

        //验证码操作,详见下文
    }
}
```

2. 创建和使用验证码

为了防止对网站的恶意攻击,现在的 Web 应用系统在登录时一般都会使用验证码。验证码在用户每次登录时由系统随机生成,以图片的形式(图片还会做得比较乱,防止程序自动识别)在登录界面上提供。用户在登录时除了输入用户名和密码之外,还需要输

入正确的验证码(每次不同)才能登录。这样可以防止恶意用户通过程序对网站进行登录攻击。

在页类中说明了两个成员变量:

```
private static string sValidator = "";   //验证字符串
private readonly string sValidatorImageUrl
    = "ValidateImage.aspx?Validator= ";
```

其中 sValidator 为静态变量,可以在页面加载之后保存验证字符串,供登录验证时使用。sValidatorImageUrl 保存为本次验证字符串生成的图片的 URL。

在页面的 Page_Load 事件中,与验证码操作有关的代码如下:

```
//如果是第一次加载页面,创建验证码
if (!Page.IsPostBack)
{
    //创建验证字符串
    sValidator = CreateValidateString(4);
    //设置验证码图片的 ImageUrl 属性
    ValidateImage.ImageUrl = sValidatorImageUrl + sValidator;
}
```

实现"创建验证字符串"函数:

```
private string CreateValidateString(int nLen)
{
    //创建一个 StringBuilder 对象
    StringBuilder sb = new StringBuilder(nLen);

    Random rnd = new Random();
    int rndi = 1000 + rnd.Next(8999);
    sb.Append(rndi.ToString());
    return(sb.ToString());
}
```

上述代码的补充说明:

- "创建验证字符串"函数只创建了一个随机的字符串,并没有为该字符串生成图片。
- 图片的生成由 ValidateImage.aspx 页面完成。

ValidateImage.aspx 页面本身无可显示内容,其处理都在 ValidateImage.aspx.cs 中完成:

```
public partial class ValidateImage : System.Web.UI.Page
{
    private readonly string ImagePath = "~/images/Validator.jpg";
    private string sValidator = "";
```

```
    private void Page_Load(object sender, System.EventArgs e)
    {
        if(Request.Params["Validator"] != null)
        {
            //获取验证字符串
            sValidator = Request.Params["Validator"].ToString();
        }

        //创建 Bmp 位图
        Bitmap bitMapImage =
            new System.Drawing.Bitmap(Server.MapPath(ImagePath));
        Graphics graphicImage = Graphics.FromImage(bitMapImage);

        //设置画笔的输出模式
        graphicImage.SmoothingMode = SmoothingMode.AntiAlias;
        //添加文本字符串
        for(int i = 0; i < sValidator.Length; i++)
        {
            graphicImage.DrawString(sValidator[i].ToString(),
                new Font("Times New Roman", 20),
                SystemBrushes.WindowText,
                new PointF(i * 25, 0));
        }

        //设置图像输出的格式
        Response.ContentType = "image/jpeg";

        //保存数据流
        bitMapImage.Save(Response.OutputStream, ImageFormat.Jpeg);

        //释放占用的资源
        graphicImage.Dispose();
        bitMapImage.Dispose();
        Response.End();
    }
}
```

3. 系统登录界面

Login.ascx 中“登录界面”部分代码如下：

```
<table width=203px bgcolor=white  cellpadding=0 cellspacing=0 class="text1">
    <tr>
        <td align=right width=30%>身份：</td>
        <td align=left width=70%>
```

```
            <asp:DropDownList ID="UserType" runat="server" Width="120">
                <asp:ListItem Value="学生">学生</asp:ListItem>
                <asp:ListItem Value="教师">教师</asp:ListItem>
                <asp:ListItem Value="管理人员">管理人员</asp:ListItem>
            </asp:DropDownList></td>
      </tr>
      <tr>
        <td align=right>用户名:</td>
        <td align=left>
          <asp:TextBox ID="UserId" Runat="server" Width="120"/>
              <font color="red" class="Normal"> * </font>
          <asp:RequiredFieldValidator id="RFVUserId" runat="server"
              ErrorMessage="<br>用户名不能为空。" ControlToValidate="UserId"
              Display="Dynamic"></asp:RequiredFieldValidator>
        </td>
      </tr>
      <tr>
        <td align=right>密码:</td>
        <td align=left>
          <asp:TextBox ID="Password" Runat="server" CssClass="InputText"
              Width="120" TextMode="Password"/><font color="red"> * </font>
          <asp:RequiredFieldValidator id="RFVPassword" runat="server"
              ErrorMessage="<br>密码不能为空。" ControlToValidate="Password"
              Display="Dynamic"></asp:RequiredFieldValidator>
        </td>
      </tr>
      <tr>
        <td align=right>验证码:</td>
        <td align=left>
          <asp:TextBox ID="Validator" Runat="server" Width="70"></asp:TextBox>
              <font color="red"> * </font>
          <asp:Image ID="ValidateImage" runat="server" Height="25px"
              Width="50px" ImageAlign="AbsBottom"/>
          <asp:RequiredFieldValidator id="rfv" runat="server"
              ErrorMessage="<br>验证码不能为空。" ControlToValidate="Validator"
              Display="Dynamic"></asp:RequiredFieldValidator>
        </td>
      </tr>
      <tr>
        <td colspan="2" align="center">
          <asp:Button ID="LoginBtn" Runat="server" Text="登录"
              CssClass="ButtonCss" Width="70px" OnClick="LoginBtn_Click">
          </asp:Button>
        </td>
```

```
    </tr>
    <tr>
        <td colspan="2" align="center"><asp:Label ID="Message" Runat="server"
            CssClass="GbText" Width="100%" ForeColor="Red"></asp:Label></td>
    </tr>
</table>
```

说明：

- 整个界面使用 Table 进行布局。
- Table 的第 1 行使用了 1 个 DropDownList 控件选择用户类型。
- 第 2 行请用户输入用户名，使用了 1 个 RequiredFieldValidator 控件进行验证。
- 第 3 行请用户输入密码。
- 第 4 行请用户输入验证码，验证图片的 ImageUrl 属性已经在 Page_Load 函数中设定。
- 第 5 行为“登录”按钮。
- 第 6 行为 1 个 Label 控件，用于显示用户名或密码有误的信息。

4. 登录处理

实现“登录”按钮的单击事件处理函数代码如下：

```
protected void LoginBtn_Click(object sender, EventArgs e)
{
    //如果页面输入不合法
    if (!Page.IsValid)
    {
        return;
    }

    //如果验证码输入错误
    if (Validator.Text != sValidator)
    {
        Message.Text = "验证码输入错误，请重新输入。";
        //重新创建验证字符串
        sValidator = CreateValidateString(4);
        //重新设置验证码图片的 ImageUrl 属性
        ValidateImage.ImageUrl = sValidatorImageUrl + sValidator;
        return;
    }

    String userId = "";
    String userName = "";

    //定义类并获取用户的登录信息
```

```
        User user = new User();

        //对用户输入进行编码
        string sUserType = Request.Params["Login1$ UserType"].ToString();
        string sUserId = Server.HtmlEncode(UserId.Text.Trim());
        string sPassword = Server.HtmlEncode(Password.Text.Trim());

        //获取用户信息
        SqlDataReader dr = user.GetUserLoginBySQL(sUserType, sUserId, sPassword);

        //判断用户是否合法
        if (dr.Read())
        {
            userId = dr["UserID"].ToString();
            userName = dr["UserName"].ToString();
        }
        dr.Close();

        //验证用户合法性,并跳转到系统平台
        if ((userId != null) && (userId != ""))
        {
            //写会话对象
            Session["UserType"] = sUserType;
            Session["UserID"] = userId;
            Session["UserName"] = userName;

            //写 Cookies
            //详见 13.5节

            //跳转到登录后的系统主页
            Response.Write(
                "<script>window.open('mainframe.aspx','_top');</script>");
        }
        else
        {
            //创建验证字符串
            sValidator = CreateValidateString(4);
            ValidateImage.ImageUrl = sValidatorImageUrl + sValidator;
            //显示错误信息
            Message.Text = "用户名或密码有误。";
        }
    }
```

说明：

- 验证控件的验证由系统自动完成，只需要通过 Page. IsValid 属性判断验证是否通过就可以了。
- 如果验证码输入有误，系统在给出提示的同时还会重新生成新的验证码，这样可以有效地防止攻击。
- 用户名与密码的正确性由 User 类的 GetUserLoginBySQL()函数进行验证。
- 同样，如果用户名与密码输入有误，系统在给出提示的同时也会重新生成新的验证码。
- 用户名与密码验证通过后，会将相关信息写入会话对象，作为系统已经登录的标志。还会写 Cookies，有关 Cookies 的详细说明见 13.5 节。加载登录后的系统主页，标志登录操作成功。

13.4 系统菜单的实现

本系统采用 TreeView 控件，从数据库中取各菜单项，动态地生成树形菜单。

menu. aspx 页面的关键代码如下：

```
<table cellpadding="0" cellspacing="0" width="100%" align="center">
    <tr>
        <td valign="top">
            <asp:TreeView ID="MenuTV" runat="server" Width="100%"
                ShowLines="True" ForeColor="Blue">
            </asp:TreeView>
        </td>
    </tr>
    <tr><td style="height: 20px;"> </td></tr>
    <tr>
        <td align="center">
            <asp:Button ID="Button1" runat="server" Text="退出登录"
                OnClick="btnExit_Click"/>
        </td>
    </tr>
</table>
```

页面使用一个 Table 控制布局。Table 的第一行包含一个 TreeView 控件，最后一行包含一个“退出登录”按钮。

TreeView 控件是 ASP. NET 中一个复杂的导航控件，具有丰富的属性和事件，本书不做详细介绍，有兴趣的读者请参考 MSDN。

下面先分析一下数据库中 MENU 表的结构及数据。

MENU 表的结构如表 13-12 所示。MENU 表的内容也是重要的，在表 13-13 当中列出。

表 13-13 菜单表(MENU)的内容

USER TYPE	TREE ID	TITLE	DESN	PARENT ID	URL	TARGET
M	0	管理人员系统	略	−1		
M	1	教师管理		0		
M	2	学生管理		0		
M	3	课程管理		0	Manage/ClassManage1. aspx	mainwork
M	4	教师管理 1		1	Manage/TeacherManage1. aspx	mainwork
M	5	教师管理 2		1	Manage/TeacherManage2. aspx	mainwork
M	6	学生管理 1		2	Manage/StudentManage1. aspx	mainwork
M	7	学生管理 2		2	Manage/StudentManage2. aspx	mainwork
M	8	管理人员列表		0		
M	9	管理人员列表 1		8	Manage/ManagerManage1. aspx	mainwork
M	10	管理人员列表 2		8	Manage/ManagerManage2. aspx	mainwork
M	11	管理人员列表 3		8	Manage/ManagerManage3. aspx	mainwork
M	12	学生管理 3		2	Manage/StudentManage3. aspx	mainwork
M	13	修改密码		0	ChangePassword. aspx	mainwork
S	0	学生系统		−1		
S	1	我的课程		0	Teacher/ClassManage2. aspx	mainwork
S	2	修改密码		0	ChangePassword. aspx	mainwork
T	0	教师系统		−1		
T	1	我的课程		0	Teacher/ClassManage2. aspx	mainwork
T	2	修改密码		0	ChangePassword. aspx	mainwork

说明：将所有菜单项都放在表 MENU 中，不同类型用户的菜单项用 UserType 字段来区分。

本系统没有进一步的权限控制，真实系统可能需要根据用户的权限控制某些菜单项是否可用，因此还需要其他与权限控制有关的字段。

同一类用户的每个菜单项具有唯一的 TreeID 号，系统规定从 0 开始使用。

每一项 ParentID 字段保存菜单父节点的 TreeID 号，这样就可在数据库中表示树形关系了。其中 ParentID 为−1 的表示为菜单根节点。

Url 字段存储完成该菜单项功能所要加载的页面。

Target 字段存储完成该菜单项功能所要加载页面的目标位置。本系统都是在 mainwork 框架（主工作区）中加载，实际的系统也可能是_blank，也可能是其他的框架。

创建树形结构的节点时，需要用到递归算法，针对每个节点递归地创建其子节点。但不必对每个节点都取一次数据库，而是一次性地将菜单数据读到一个 DataTable 对象

中，然后再使用DataTable的Select方法来取各节点的子节点。相关代码在menu.aspx.cs中：

```
protected void Page_Load(object sender, EventArgs e)
{
    //如果未登录，则不调入菜单
    if (Session["UserID"] == null)
    {
        return;
    }

    if (!Page.IsPostBack)
    {   //显示菜单数据
        BindCategoryTreeView(MenuTV);
    }
}

/// <summary>
/// 加载菜单数据
/// </summary>
/// <param name="treeView">作为菜单的TreeView控件</param>
public void BindCategoryTreeView(TreeView treeView)
{
    //从MENU表中取当前类型用户的所有菜单项
    Menu mn = new Menu();
    DataTable dataTable =
        mn.GetMenuItems(Session["UserType"].ToString());

    //清空菜单TreeView的所有节点
    treeView.Nodes.Clear();

    //取得根节点项数据
    DataRow[] rowList = dataTable.Select("ParentID= '-1'");
    if (rowList.Length <= 0) return;

    //创建根节点
    TreeNode rootNode = new TreeNode();
    //设置根节点属性
    rootNode.Text = rowList[0]["Title"].ToString();
    rootNode.Value = rowList[0]["TreeID"].ToString();
    rootNode.Expanded = True;
    rootNode.NavigateUrl = "menu.aspx";
    rootNode.Target = "menu_iframe";
```

```
        //将根节点添加到菜单 TreeView
        TreeView.Nodes.Add(rootNode);

        //创建根节点的所有子节点
        CreateChildNode(rootNode, dataTable);
        dataTable.Clear();
    }

    /// <summary>
    /// 递归函数,生成某个节点的所有子节点
    /// </summary>
    /// <param name="parentNode">父节点</param>
    /// <param name="dataTable">保存菜单数据的 DataTable 对象</param>
    private void CreateChildNode(TreeNode parentNode, DataTable dataTable)
    {
        //设置选择条件,取当前父节点的所有子节点
        DataRow[] rowList = dataTable.Select("ParentID= '" + parentNode.Value+ "'");
        //对每个子节点进行操作
        foreach (DataRow row in rowList)
        {   //创建新节点
            TreeNode node = new TreeNode();
            //设置子节点的属性
            node.Text = row["Title"].ToString();
            node.Value = row["TreeID"].ToString();
            node.Expanded = True;
            if (row["Url"].ToString() != "")
            {
                node.NavigateUrl = row["Url"].ToString();
                node.Target = row["Target"].ToString();
            }
            else
            {
                node.NavigateUrl = "menu.aspx";
                node.Target = "menu_iframe";
            }
            //将新节点添加到菜单 TreeView
            //作为父节点的子节点
            parentNode.ChildNodes.Add(node);
            //递归调用,生成该子节点的所有子节点
            CreateChildNode(node, dataTable);
        }
    }
```

其中,GetMenuItems()函数从数据库中取得目录数据,在 Menu 类中定义,代码如下:

```
/// <summary>
/// 取得目录数据
/// </summary>
/// <param name="sUserType">用户类型</param>
/// <returns>保存目录项的 DataTable</returns>
public DataTable GetMenuItems(string sUserType)
{
    //生成 SQL 语句
    string cmdText = "SELECT *  FROM menu where UserType='";
    switch (sUserType)
    {
        case "教师":
            cmdText = cmdText + "T";
            break;
        case "学生":
            cmdText = cmdText + "S";
            break;
        case "管理人员":
            cmdText = cmdText + "M";
            break;
        default:
            return null;
    }
    cmdText = cmdText + "'";

    //取数据
    Database db = new Database();
    DataTable dt = db.RunSQLtoDataTable(cmdText);

    //返回 DataTable
    return dt;
}
```

系统的退出登录功能也在此页面上实现。为“退出登录”按钮编写单击事件处理函数代码如下：

```
/// <summary>
/// “退出登录”按钮处理事件
/// </summary>
protected void btnExit_Click(object sender, EventArgs e)
{
    //清除 Cookie(详见 13.5 节)
    Response.Cookies["userInfo"].Expires = DateTime.Now.AddDays(-1);
    //清除会话对象
    Session.Clear();
```

```
    //加载带有登录界面的系统主页
    Response.Redirect("rightframe.aspx");
}
```

13.5 Cookie的使用

前面已经介绍了系统登录(13.3节)和系统退出(13.4节)的实现方法。无论对于教师、学生还是管理人员,本系统都将是最常用的一个业务系统。如果每次想进入系统时都需要登录,则稍嫌繁琐。使用Cookie允许用户在一次登录后的一段时间内不必再登录,而是在加载主页时按照上次登录的结果,直接进入系统。

Cookie提供了一种在Web应用程序中存储用户特定信息的方法。本节就介绍Cookie的有关概念及使用方法。

13.5.1 什么是Cookie

通俗地说,Cookie是一小段文本信息。它随着用户请求和页面在Web服务器和浏览器之间传递,Cookie所包含的信息每次用户访问站点时Web应用程序都可以读取。

Cookie在Web服务器和浏览器之间的传递过程如下:

(1) Web服务器将用户请求的页面发回给用户时,除页面之外,还可以包含一个Cookie。

(2) 用户的浏览器在获得页面的同时还获得了该Cookie,并将它存储在用户硬盘上的某个文件夹中。

(3) 以后,如果该用户再次请求相同的URL时,浏览器便会在本地硬盘上查找与该URL关联的Cookie。如果该Cookie存在,浏览器便将该Cookie与页面请求一起发送到Web服务器。

(4) Web服务器可以从该Cookie读取信息,并进行相应的处理。

Cookie与网站关联,而不是与特定的页面关联。因此,无论用户请求站点中的哪一个页面,浏览器和服务器都将交换Cookie信息。用户访问不同站点时,各个站点都可能会向用户的浏览器发送一个Cookie,浏览器会分别存储所有Cookie。

HTTP是一种无连接的协议,除短暂的实际交换信息的时间外,浏览器和Web服务器间都是断开连接的。Cookie提供了一种保持Web应用程序连续性(即执行状态管理)的方法,最常用的就是Cookie能帮助网站存储有关访问者的信息。

大多数浏览器支持最大为4096字节的Cookie。由于这个限制,最好用Cookie来存储少量数据,如用户ID等。

浏览器还限制站点可以在用户计算机上存储的Cookie的数量。大多数浏览器只允许每个站点存储20个Cookie。

13.5.2 写入Cookie

浏览器负责管理用户系统上的Cookie。Cookie通过HttpResponse对象发送到浏览

器,该对象相关的公开属性是称为 Cookies 的集合。要发送给浏览器的所有 Cookie 都必须添加到此集合中。创建 Cookie 时,需要指定 Name 和 Value。每个 Cookie 必须有一个唯一的名称,以便以后从浏览器读取 Cookie 时可以识别它。

还可以设置 Cookie 的到期日期和时间(Expires 属性)。用户访问处理 Cookie 的网站时,浏览器将删除过期的 Cookie。

注意:用户可随时清除其计算机上的 Cookie。即便存储的 Cookie 还未到期。

下面代码使用了两种方法,分别将两个 Cookie 添加到 Response 的 Cookies 集合中:

```
Response.Cookies["UserType"].Value = "学生";
Response.Cookies["UserType"].Expires = DateTime.Now.AddDays(1);

HttpCookie aCookie = new HttpCookie("UserID");
aCookie.Value = "S201";
aCookie.Expires = DateTime.Now.AddDays(1);
Response.Cookies.Add(aCookie);
```

在上面的例子中,两个 Cookie 的到期时间都是一天之后,但要分别设置。在许多实际应用中,有些需要放在 Cookie 中的信息是相互关联的,具有相同的到期时间,则可以使用多值 Cookie。

可以在一个 Cookie(有一个名称)中存储多个名称/值对,这样的名称/值对称为子键,这样的 Cookie 称为多值 Cookie。如本系统中保存用户信息的 Cookie 就是一个多值 Cookie。

在本系统的登录处理函数中,如果用户登录信息验证通过,在写会话对象之后,还会写 Cookie,代码如下:

```
//写 Cookies
Response.Cookies["userInfo"]["UserType"] = sUserType;
Response.Cookies["userInfo"]["UserID"] = userId;
Response.Cookies["userInfo"]["UserName"] = userName;
Response.Cookies["userInfo"].Expires = DateTime.Now.AddDays(1);
```

13.5.3 读取 Cookie

浏览器向 Web 服务器发出请求时,会随请求一起发送该服务器的 Cookie。在 ASP.NET 中,可以在服务器端使用 HttpRequest 对象读取 Cookie,读取方式与将 Cookie 写入 HttpResponse 对象的方式基本相同。

下面的代码分别使用两种方法,将上面写入的 Cookie 读出,并将其值显示在 Label 控件中:

```
if(Request.Cookies["UserType"] != null)
    Label1.Text = Server.HtmlEncode(Request.Cookies["UserType"].Value);

if(Request.Cookies["UserID"] != null)
```

```
{
    HttpCookie aCookie = Request.Cookies["UserID"];
    Label2.Text = Server.HtmlEncode(aCookie.Value);
}
```

说明：在尝试获取Cookie的值之前，应确保该Cookie存在。显示Cookie的内容前，最好先调用HtmlEncode方法对Cookie的内容进行编码，这样可以确保恶意用户没有向Cookie中添加可执行脚本。

读取多值Cookie的方法也与设置方法类似。在本系统默认主页的隐藏代码中，包含如下代码：

```
//检查Cookies对象
if (Request.Cookies["userInfo"] != null)
{
    Session["UserType"]
        = Server.HtmlEncode(Request.Cookies["userInfo"]["UserType"]);
    Session["UserID"]
        = Server.HtmlEncode(Request.Cookies["userInfo"]["UserID"]);
    Session["UserName"]
        = Server.HtmlEncode(Request.Cookies["userInfo"]["UserName"]);
    }
```

说明：上述代码检查Cookies中"userInfo"是否存在。如存在，则将其内容写入到会话对象中。前文已经介绍过，本系统通过检查会话对象中的用户信息来判断系统是否已经登录。这样，本段代码与前面的代码相结合，能够使用户在一次登录后的一天之内不必重新登录，就能够直接进入系统。

13.5.4 删除Cookie

Cookie存储在用户的计算机中，用户可以使用浏览器功能删除本地的Cookie。服务器无法直接删除用户端的Cookie，但可以通过浏览器来删除Cookie。方法是将所要删除的Cookie的到期日期设置为早于当前日期的某个日期，当浏览器检查Cookie的到期日期时，便会丢弃这个已过期的Cookie。

本系统菜单页的"退出登录"处理函数中，包含如下代码：

```
//清除Cookie
Response.Cookies["userInfo"].Expires = DateTime.Now.AddDays(-1);
//清除会话对象
Session.Clear();
```

上面代码先将Cookies中"userInfo"置为过期，再清除会话对象，这样，当用户再想进入系统时，就需要重新登录了。

13.6 修改口令

修改口令是任何系统都会具有的一个功能，在本系统中，它是三类用户都要使用的一个通用模块，界面如图 13-7 所示。

用户名:	m
旧密码:	
新密码:	
确认密码:	
	修改

图 13-7 修改口令界面

在 ChangePassword.aspx 中，关键部分代码如下：

```
<table class="text" style="BORDER-COLLAPSE: collapse;"
        borderColor="#5179BB" width="30%" border="1">
    <tr>
        <td style="width:30%" align="right">用户名:</td>
        <td><asp:textbox id="UserID" runat="server" Width="95%"
            Enabled="False"></asp:textbox></td>
    </tr>
    <tr>
        <td align="right">旧密码:</td>
        <td><asp:textbox id="OldPassword" runat="server" Width="95%"
                TextMode="Password"></asp:textbox>
            <asp:RequiredFieldValidator id="rfO" runat="server"
                ErrorMessage="密码不能为空。"
                ControlToValidate="OldPassword" Display="Dynamic">
            </asp:RequiredFieldValidator>
        </td>
    </tr>
    <tr>
        <td align="right">新密码:</td>
        <td><asp:textbox id="NewPassword" runat="server" Width="95%"
                TextMode="Password"></asp:textbox>
            <asp:RequiredFieldValidator id="rfN" runat="server"
                ErrorMessage="密码不能为空。"
                ControlToValidate="NewPassword" Display="Dynamic">
            </asp:RequiredFieldValidator>
        </td>
    </tr>
    <tr>
```

```
            <td align="right">确认密码:</td>
            <td><asp:textbox id="PasswordStr" runat="server" Width="95%"
                    TextMode="Password"></asp:textbox>
              <asp:RequiredFieldValidator id="rfS" runat="server"
                    ErrorMessage="密码不能为空。"
                    ControlToValidate="PasswordStr" Display="Dynamic">
              </asp:RequiredFieldValidator>
            </td>
        </tr>
        <tr>
            <td></td>
            <td><asp:CompareValidator ID="CompareValidator1" runat="server"
                    ErrorMessage="两次输入的密码不相同。"
                    ControlToCompare="NewPassword"
                    ControlToValidate="PasswordStr">
              </asp:CompareValidator>
            </td>
        </tr>
        <tr>
            <td></td>
            <td align="center"><asp:Button ID="UpdateBtn" runat="server"
                Text="修改" Width="100px" OnClick="UpdateBtn_Click"/></td>
        </tr>
    </table>
    <asp:Label ID="lbMess" runat="server" Text=""
        ForeColor="Red">
    </asp:Label>
```

可以看出,大部分代码与登录界面的代码相似。所不同的是,在确认密码处使用了CompareValidator验证控件来验证两次输入的新密码是否一致。

在"修改"按钮的处理函数中,旧密码的验证部分也与登录处理函数相似,但多了调用User类的ChangeUserPwd函数修改数据库的过程。相关程序这里不再介绍,请有兴趣的读者自己完成。

13.7 教师管理

本系统提供了两种教师管理的方法。第一种方法使用DetailsView控件,在8.4节中已经介绍。第二种方法主要是使用FormView控件,其操作界面如图13-8所示。其中,对DropDownList控件(界面的上部)进行数据绑定的内容在9.2节中已经介绍,本节简单介绍一下对FormView控件的使用。



图 13-8 教师管理界面(一)

FormView 是 ASP. NET 2. 0 中的新增控件。与 DetailsView 控件一样,可对单条数据记录进行操作。使用 FormView 控件可以编辑、删除和插入记录。

与 DetailsView 控件不同的是,FormView 控件使用用户定义的模板,在模板中使用数据绑定表达式来显示字段的值,而不是使用 asp:BoundField 进行字段绑定。使用模板可以更灵活地控制数据的显示方式。

FormView 控件可以创建的模板包括 EditItemTemplate、EmptyDataTemplate、FooterTemplate、HeaderTemplate、ItemTemplate、InsertItemTemplate 和 PagerTemplate。

创建一个名为 UseFormView 的网站。

添加现有项,将 ControlBind 网站的 Web. Config 文件添加到本网站,ControlBind 网站在 9. 2 节中创建。

创建一个名为 TeacherManage2. aspx 的新页面,将 ControlBind 网站的 TeacherManage2. aspx 页面的相关代码复制过来,包括 h1、DropDownList1 和 SqlDataSource1,并将 DropDownList1 的 AutoPostBack 属性改为 True,页面现在就能执行。

从工具箱中拖一个 FormView 到页面上来,初始代码如下:

```
<asp:FormView ID="FormView1" runat="server">
</asp:FormView>
```

在设计视图中为 FormView1 新建数据源:选择 TEACHER 表的所有字段;生成 INSERT、UPDATE 和 DELETE 语句;生成 WHERE 子句,字段为 USERID,源为 DropDownList1 控件。

再到源视图中查看代码,发现 FormView1 的代码已经发生了很大的变化。主要是增加了＜EditItemTemplate＞、＜InsertItemTemplate＞和＜ItemTemplate＞模板。执行页面,效果如图 13-9 所示。



图 13-9 教师管理界面(二)

当在上部的 DropDownList 中选择不同的教师时,下面的信息也会随之改变。

有兴趣的读者可以参照本书应用实例中的代码继续完成本功能,主要是对上述各模板进行修改,再套用合适的格式,以便得到更好的显示效果。

13.8 学生管理

本系统提供了三个学生管理的功能，分别用不同的方法实现。前两种方法使用 GridView 控件，在 8.2 节已经介绍，现在来介绍第三种方法。

很多实际应用中需要对记录数比较多的信息进行管理，如学生信息。由于记录数多，有的甚至达到十万、百万条，直接列表效果不好，因此对这类信息的管理往往是以查询为基础的。本功能将查询界面与管理界面结合在一起，能够极大地方便用户的使用。

本系统学生管理 3 的操作界面如图 13-10 所示。

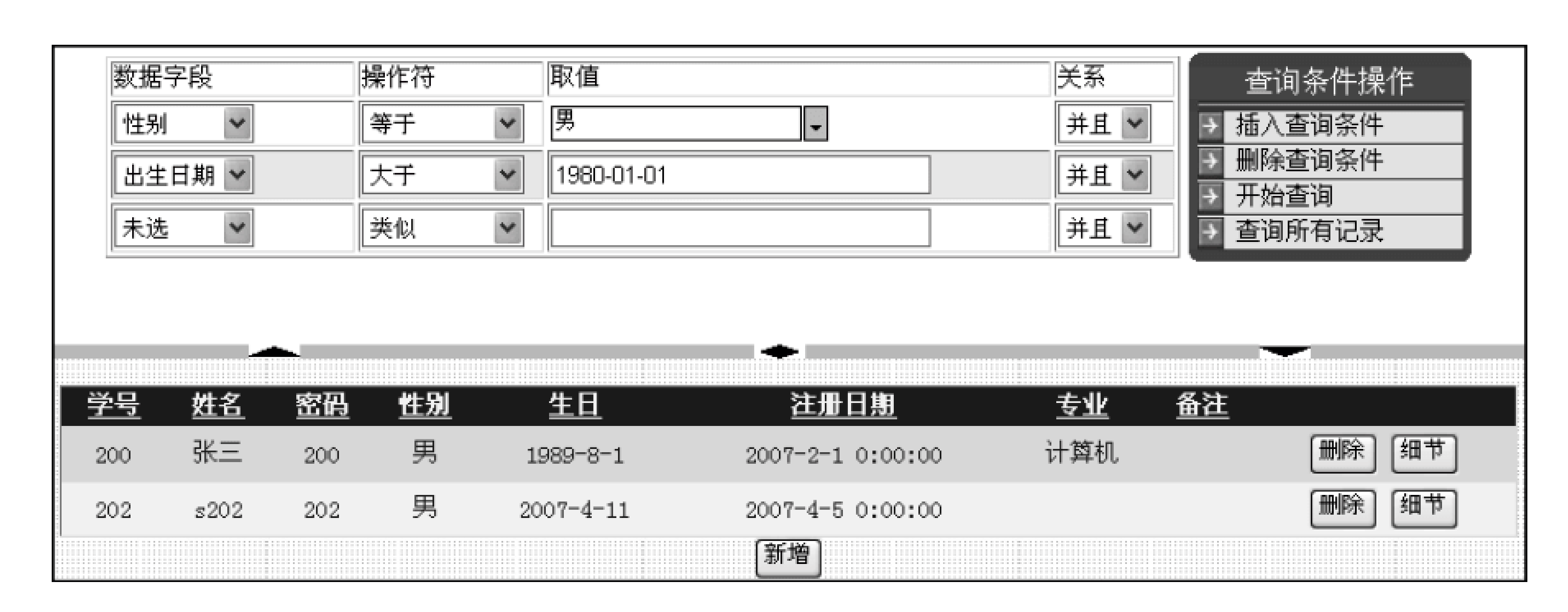


图 13-10 学生管理界面

说明：

- 界面主要分为上下两部分：上部为查询界面，下部为管理界面。
- 查询界面包括左侧的查询条件列表和右侧的查询条件操作部分。
- 在查询条件列表中，可以选择对任意数据字段字义查询条件，可以定义多个查询条件(每个一行)，可以定义各条件之间的关系(“并且”或“或者”)。
- 查询条件操作部分可以对查询条件进行插入和删除操作，可以按当前定义的查询条件进行查询，可以查询所有记录(忽略查询条件)。
- 管理界面主要是学生信息的列表，仍然使用 GridView 控件实现，借助其内置的功能完成排序、分页、删除、选择等操作。
- 按“新增”按钮添加新记录。
- 在上下两部分之间有一个分隔条，上面有三个图形按钮，单击其中一个可以使两部分中的一部分占满全屏或重新分区。

创建一个名为 StudentManage3.aspx 的新页面，页面使用框架，将主工作区分为上中下三个部分，简化后的代码如下：

```
<frameset rows="*,8,66%" name="SpliterFrameSet">
    <frame name="QryFrame" src="QrySet.aspx?TableName=STUDENT">
    <frame name="SpliterFrame" src="mainSpliter.aspx">
    <frame name="QryResultFrame" src="QryResult.aspx">
</frameset>
```

创建 QrySet.aspx 页面，完成查询条件的定义。

在一个完整的应用系统中，会有多个与本功能类似的管理功能，因此，将 QrySet.aspx 页面做成一个通用界面，可以为所有类似的功能服务。所以，在加载该页面时需要向其传递一个参数 TableName，指明所要管理的数据库表名。

QrySet.aspx 是一个应用了很多编程技巧的页面，尤其是大量使用了 JavaScript 程序。由于 JavaScript 不是本书的重点，为了便于教学，突出重点，对该功能做了较大的简化。只允许定义一个查询条件，“取值”栏目不再支持列表选择。

在查询条件列表部分，“数据字段”栏目中是一个下拉式列表，其中列出了当前所要管理的数据库表的所有可进行查询条件定义的字段和左、右括号。所谓“可进行查询条件定义的字段”可以是该表的全部字段，也可以是部分字段(根据实际需要)。这些字段的信息存储在数据库表 SYS_FIELD 中，在页面加载时，列表在服务器端动态生成。

“操作符”栏目包括 SQL 语句的条件子句中所有可能出现的操作符。列表时列出的是中文操作符名称，在生成 SQL 语句时会将其转换为真正的操作符字符。

“取值”栏目用于输入查询条件取值。对于某些取值可枚举的字段，“取值”也可进行列表选择。哪些字段进行列表选择由系统根据 SYS_FIELD 表中的字段信息判定。

当有多个查询条件时，其次序是重要的。在右侧的查询操作界面中按“插入查询条件”按钮，在当前的查询条件前插入一个新的查询条件；按“删除查询条件”删除当前的查询条件。

单击“开始查询”按钮，触发 SubmitQry()函数。函数扫描查询条件列表中的各个查询条件定义，并将其组合成一个完整的查询条件子句。然后将生成的查询条件子句置入名为 conditionForm 的 Form 的 qrycondition 域中，提交该 Form。conditionForm 的定义代码如下：

```
<form action="QryResult.aspx" name="conditionForm" method="post"
        target="QryResultFrame">
    <input type="hidden" name="qrycondition" value="">
</form>
```

单击“查询所有记录”按钮时触发 queryall()函数。其功能是将 conditionForm 的 qrycondition 域置空，然后提交该 Form。

从上文可以看出，查询结果由页面 QryResult.aspx 进行处理。QryResult.aspx 仍然使用 GridView 控件实现，这样就可以借助其内置的功能完成排序、分页、删除、选择等操作。

页面加载时，在 Page_Load()函数中根据 qrycondition 参数的值，生成新的 SQL 语句，并相应地改变 SqlDataSource1 的 SelectCommand 属性值。

单击“删除”按钮，利用 GridView 控件的内置功能删除记录。

单击“细节”和“新增”按钮，都是在新的浏览器窗口中打开 StudentDetail3.aspx 页

面。单击“细节”按钮时，会向 StudentDetail3. aspx 页面传递一个 USERID；单击“新增”按钮时则不传递参数，StudentDetail3. aspx 页面根据是否有传入的 USERID 来判断是进行记录的修改还是新增。

StudentDetail3. aspx 页面中所采用的大多数技术在前文中都介绍过，这里不再赘述。其中有特色的地方就是采用 ASP. NET 的日历控件制作了弹出式的日期输入控件，有兴趣的读者可以参考系统代码进行研究。

13.9 课程管理

“课程”是网络学院中一个核心概念，学生、教师和管理人员都要对课程进行操作。

本系统采用的是集中管理机制，即学生自己不能自由选课，哪个学生选哪些课都由管理人员来确定。因此，在为管理人员实现的课程管理功能中，除了对课程基本信息的管理之外，还包括为课程选择任课教师和选课学生的功能。

ClassManage1. aspx 页面完成管理人员的课程管理功能，如图 13-11 所示。

课程管理1

编号	课程名	开课时间	专业	备注	
J01	C语言程序设计	2007-3-24 18:33:52	计算机		修改
J02	数据结构	2007-3-24 19:15:02	计算机		修改
J03	网络程序设计	2007-3-24 19:15:31	计算机		修改
J04	.NET程序设计	2007-3-24 19:16:00	计算机		修改

新增

编号	J03	课程名	网络程序设计	开课时间	2007-3-24 19:15:31
专业	计算机	备注			

保存 取消 删除 任课教师 选课学生

图 13-11 课程管理界面

本功能仍然采用 GridView 控件显示课程列表，利用其内置的功能完成排序、选择等操作。

页面下部的细节信息管理界面一般情况下是隐藏的，当“新增”或“修改(选择)”时才显示。其实现方法与 8.2.5 小节中“学生管理 2”的实现方法相同，这里不再赘述。

在“修改”课程信息时，细节界面中包括两个超链接(“新增”时没有)——任课教师和选课学生。单击选课学生进入当前课程的选课学生管理界面，如图 13-12 所示。

界面中列出了选修当前课程的所有学生，可以删除，也可以增加新的学生。注意，本功能只是示例功能，并不实用。因为学生数量也许很多，在新增时应该有配合的查询功能(如使用一个弹出式窗口)，方便学生编号的选择，有兴趣的读者可以自行完成此功能。

任课教师功能的实现方式与选课学生相同。

选课学生

学生编号	课程编号	选课时间	
204	J03	2007-3-24 19:33:32	删除
205	J03	2007-3-24 19:33:32	删除
206	J03	2007-5-18 18:00:00	删除
207	J03	2007-3-24 19:33:32	删除
208	J03	2007-3-24 19:33:32	删除

学生编号
选课时间
新增
返回

图 13-12 选课学生界面

13.10 我的课程

学生和教师都不必关心所有的课程，只需要关心自己所选、所教的课程就可以了。

ClassManage2.aspx 页面同时供学生和教师使用，根据用户类型的不同，可能屏蔽某些功能。

本功能仍然采用 GridView 控件显示课程列表，但在 Page_Load()函数中会根据用户类型（学生 or 教师）对 SQL 语句和界面做动态的修改。单击某门课程的“进入”按钮，可进入该课程的细节信息管理界面 ClassDetail1.aspx，如图 13-13 所示。

课程名称：网络程序设计

课号	J03	开课时间	2007-3-24 19:15:31	专业	计算机
备注					

课件列表

课件名称	文件类型	课件说明	操作
a	application/octet-stream	aaaaaaaaaaaaaaa	删除
textfile	text/plain	测试文本文件	删除

上传课件

课件名称：
课件内容：浏览...
课件说明：
上传课件

其他操作

问题与解答

图 13-13 课程细节管理界面

ClassDetail1.aspx 页面实现课程细节的管理。课程细节管理界面从上到下分为几个部分。

最上面的部分显示课程基本信息，采用 FormView 控件实现。因为只显示信息，不需要修改，所以只使用了<ItemTemplate>模板。

课件列表用DataGrid控件实现。DataGrid控件是老版本的Visual Studio所支持的一个控件，在本书的8.2.1节中曾经简单介绍过，在VS2005中已经由GridView控件所代替。在此功能中使用DataGrid控件，供有兴趣的读者对两个控件进行比较。当教师使用此功能时，可以通过“删除”超链接删除相应的课件，而学生则无此权限。

上传课件部分采用FileUpload控件实现。FileUpload控件的使用在5.2节已经详细介绍。与5.2节不同，本功能在存储上传文件的同时还会将文件的相关信息写到数据库中的FILES表中，供课件列表时使用。另外，当删除一个课件时，除了删除数据库中的内容外，还要根据数据库中的信息删除相应的操作系统文件。

单击“问题与解答”超链，进入当前课程的问题与解答功能。“问题与解答”的实现方法在9.4节已经详细介绍，这里不再赘述。

最后还得再一次承认，出于教学的目的，本系统不是一个实用的系统。但如果已经掌握了本书所介绍的主要内容，将本系统现有功能改得更加实用是很容易的，那也将是一个很有意思的过程。

本书作者一向认为，编程不是“学”会的，而是“编”会的。因此，学习完本书的内容之后，如果有机会很快将其应用于开发实践当然是最好的；如果一时还没有这样的机会，改写本书的实例系统也不失为一个进一步深化所学内容的选择。

如果要建立一个真正的网络学院，也可以在本书应用实例的基础上开始工作。例如，可以为教师系统增加查看选课学生信息、制订教学计划、留作业与批改、考试等功能；为学生系统增加个人信息管理、选课申请与审批等功能；课件管理也可以增加分类、自动播放等功能；等等。对于一个初学者来说，设计并实现所有这些功能或是其中的一小部分，哪怕仅仅是想一想如何实现，也会极大地提高Web应用程序的设计与开发能力。

程序的设计与开发是一个创造性的过程，如果读者也能从中感受到快乐，那么完全可以选择它作为终身职业。

习　　题

1. 参照本章介绍的畅想网络学院的内容，在计算机上建立好示例数据库，完成程序代码，并按照10.4节介绍的方法对代码进行部署，观察网站的执行效果。

2. 请用FormView控件实现畅想网络学院中的“学生管理3”的细节显示功能。

3. 请自行完成畅想网络学院中修改密码部分的代码。

4. 请自行完成13.7节中介绍的“教师管理2”部分的代码。

5. 在选课学生管理中，新增学生的时候，如何实现方便的查询？

6. 为学生增加选课申请功能。提示：增加学生填写选课申请单的界面，将学生填好的申请存入数据库等待处理。学生要能够查询自己的选课申请的处理情况。

7. 为管理人员增加处理选课申请的功能。提示：对数据库中的学生选课申请进行列表，可逐项选择进行处理。

参 考 文 献

[1] Jesse Liberty，Dan Hurwitz 著. Programming ASP.NET 中文版. 瞿杰，赵立东，张昊译. 北京：电子工业出版社，2007

[2] 李玉林，王岩. ASP.NET 2.0 网络编程从入门到精通. 北京：清华大学出版社，2006

[3] 郑耀东，蔡骞. ASP.NET 网络数据库开发实例精解. 北京：清华大学出版社，2006

[4] 周峰. 白领就业指南——网络编程案例教学. 北京：电子工业出版社，2007

[5] 谢希仁. 计算机网络(第四版). 大连：大连理工大学出版社，2004

[6] David Flanagan 著. JavaScript 权威指南. 张铭泽等译. 北京：机械工业出版社，2006

[7] 吴晨，陈建孝. C# 网络与通信程序设计案例精讲. 北京：清华大学出版社，2006

读者意见反馈

亲爱的读者：

感谢您一直以来对清华版计算机教材的支持和爱护。为了今后为您提供更优秀的教材，请您抽出宝贵的时间来填写下面的意见反馈表，以便我们更好地对本教材做进一步改进。同时如果您在使用本教材的过程中遇到了什么问题，或者有什么好的建议，也请您来信告诉我们。

地址：北京市海淀区双清路学研大厦 A 座 602　计算机与信息分社营销室 收

邮编：100084　电子邮件：jsjjc@tup.tsinghua.edu.cn

电话：010-62770175-4608/4409　邮购电话：010-62786544

教材名称：Web 应用开发技术

ISBN：978-7-302-17671-8

个人资料

姓名：________ 年龄：____ 所在院校/专业：____________

文化程度：______ 通信地址：____________________

联系电话：______ 电子信箱：____________________

您使用本书是作为：□指定教材 □选用教材 □辅导教材 □自学教材

您对本书封面设计的满意度：

□很满意 □满意 □一般 □不满意　改进建议________________

您对本书印刷质量的满意度：

□很满意 □满意 □一般 □不满意　改进建议________________

您对本书的总体满意度：

从语言质量角度看 □很满意 □满意 □一般 □不满意

从科技含量角度看 □很满意 □满意 □一般 □不满意

本书最令您满意的是：

□指导明确 □内容充实 □讲解详尽 □实例丰富

您认为本书在哪些地方应进行修改？（可附页）

____

____

您希望本书在哪些方面进行改进？（可附页）

____

____

电子教案支持

敬爱的教师：

为了配合本课程的教学需要，本教材配有配套的电子教案（素材），有需求的教师可以与我们联系，我们将向使用本教材进行教学的教师免费赠送电子教案（素材），希望有助于教学活动的开展。相关信息请拨打电话 010-62776969 或发送电子邮件至 jsjjc@tup.tsinghua.edu.cn 咨询，也可以到清华大学出版社主页（http://www.tup.com.cn 或 http://www.tup.tsinghua.edu.cn）上查询。

高等院校信息技术规划教材

系 列 书 目

书　名	书　号	作　者
数字电路逻辑设计	978-7-302-12235-7	朱正伟　等
计算机网络基础	978-7-302-12236-4	符彦惟　等
微机接口与应用	978-7-302-12234-0	王正洪　等
XML 应用教程(第 2 版)	978-7-302-14886-9	吴　洁
算法与数据结构	978-7-302-11865-7	宁正元　等
算法与数据结构习题精解和实验指导	978-7-302-14803-6	宁正元　等
工业组态软件实用技术	978-7-302-11500-7	龚运新　等
MATLAB 语言及其在电子信息工程中的应用	978-7-302-10347-9	王洪元
微型计算机组装与系统维护	978-7-302-09826-3	厉荣卫　等
嵌入式系统设计原理及应用	978-7-302-09638-2	符意德
C++ 语言程序设计	978-7-302-09636-8	袁启昌　等
计算机信息技术教程	978-7-302-09961-1	唐　全　等
计算机信息技术实验教程	978-7-302-12416-0	唐　全　等
Visual Basic 程序设计	978-7-302-13602-6	白康生　等
单片机 C 语言开发技术	978-7-302-13508-1	龚运新
ATMEL 新型 AT89S52 系列单片机及其应用	978-7-302-09460-8	孙育才
计算机信息技术基础	978-7-302-10761-3	沈孟涛
计算机信息技术基础实验	978-7-302-13889-1	沈孟涛　著
C 语言程序设计	978-7-302-11103-0	徐连信
C 语言程序设计习题解答与实验指导	978-7-302-11102-3	徐连信　等
计算机组成原理实用教程	978-7-302-13509-8	王万生
微机原理与汇编语言实用教程	978-7-302-13417-6	方立友
微机组装与维护用教程	978-7-302-13550-0	徐世宏
计算机网络技术及应用	978-7-302-14612-4	沈鑫剡　等
微型计算机原理与接口技术	978-7-302-14195-2	孙力娟　等
基于 MATLAB 的计算机图形与动画技术	978-7-302-14954-5	于万波
基于 MATLAB 的信号与系统实验指导	978-7-302-15251-4	甘俊英　等
信号与系统学习指导和习题解析	978-7-302-15191-3	甘俊英　等
计算机与网络安全实用技术	978-7-302-15174-6	杨云江　等
Visual Basic 程序设计学习和实验指导	978-7-302-15948-3	白康生　等
Photoshop 图像处理实用教程	978-7-302-15762-5	袁启昌　等
数据库与 SQL Server 2005 教程	978-7-302-15841-7	钱雪忠　著
计算机网络实用教程	978-7-302-16212-4	陈　康　等